**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

*Факультет інформатики та обчислювальної техніки*

*Кафедра обчислювальної техніки*

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
“___”_____20__ р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія програмного забезпечення комп’ютерних систем»**

**спеціальності 121 «Інженерія програмного забезпечення»**

**на тему: «Веб сервіс пошуку оренди житла з покращеною системою ідентифікації особистості»**

Виконав:
студент _IV_курсу, групи__ІП-62__
Федорович Вадим Романович _____
(підпис)

Керівник:
Доцент, кандидат технічних наук
Роковий Олександр Петрович _____
(підпис)

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____
(підпис)

Рецензент
Доцент, кандидат технічних наук
Коган Алла Вікторівна _____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**ІМ. ІГОРЯ СІКОРСЬКОГО»**

Кафедра обчислювальної техніки
Напрям підготовки - 6.050103 - «Програмна інженерія»

Затверджую:
зав. кафедрою
_____

«____»_____2020

# ЗАВДАННЯ

на бакалаврську атестаційну роботу студента

Федоровича Вадима Романовича

1. Тема роботи «*Веб сервіс пошуку оренди житла з покращеною системою ідентифікації особистості*», керівник проєкту Роковий Олександр Петрович, доцент, кандидат технічних наук,

затверджена наказом по університету від «__» ___ 2020р. №_____

2. Термін здачі студентом закінченого роботи _____*.2020р.*
3. Вихідні дані до роботи *технічне завдання, теоретичні данні.*
4. Зміст розрахунково-пояснювальної записки: *опис предметної області, дослідження веб-сервісів пошуку оренди житла, веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості.*

5. Консультанти розділів проекту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| нормоконтроль | д.т.н., проф. Сімоненко В. П. | | |
| | | | |
| | | | |

6. Дата видачі завдання _____

Календарний план

| № з/п | Назва етапів виконання дипломного проекту | Термін виконання етапів проекту | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 15.09.2019 - 10.02.2020 | |
| 2. | Вивчення та аналіз завдання | 11.02.2020 - 15.04.2020 | |
| 3. | Написання вступної частини та огляду предметної області | 16.04.2020 - 31.04.2020 | |
| 4. | Розробка архітектури системи | 01.05.2020 - 11.05.2020 | |
| 5. | Програмна реалізація системи | 12.05.2020 - 26.05.2020 | |
| 6. | Оформлення документації дипломної роботи | 27.05.2020 - 31.05.2020 | |
| 7. | Передзахист | | |
| 8. | Захист | | |

Студент _____ Вадим ФЕДОРОВИЧ
(підпис)

Керівник проекту _____ Олександр РОКОВИЙ
(підпис)

# АНОТАЦІЯ

В даному бакалаврському дипломному проекті був реалізований веб-сервіс пошуку оренди житла з можливістю авторизації за допомогою системи BankID. Веб-сервіс надає функціонал пошуку житла за визначеними фільтрами, а також візуалізує результати пошуку на карті у вигляді маркерів з пріоритетністю по їх місцезнаходженню.

Програмний продукт був створений на мові C# з використанням фреймворку .NET Core та .NET Standart у середовищі програмування Visual Studio 2019. Також були застосовані доповнення до Visual Studio, такі як: Bundler & Minifier, GitLab Extension for Visual Studio. Для контролю версії проекту було застосовано систему Git та систему керування репозиторіями програмного коду GitLab.

Дана реалізація веб-сервісу була побудована на основі вже існуючих аналогів. Враховані усі їх недоліки, зокрема була розроблена краща система безпеки ідентифікації особистості, а також додані інноваційні рішення відображення результатів пошуку на карті.

Ключові слова: веб-сервіс пошуку оренди житла, безпека, BankID.

Розмір пояснювальної записки – 60 аркушів, містить 30 ілюстрацій, 5 додатків.

# ANNOTATION

In this bachelor's degree project, a web service for finding rental housing with the possibility of authorization using the BankID system was implemented. The web service provides housing search functionality by certain filters, as well as visualizes the search results on the map in the form of markers with priority according to their location.

The software product was created in C# using the .NET Core and .NET Standart frameworks in the Visual Studio 2019 programming environment. Additions to Visual Studio, such as: Bundler & Minifier, GitLab Extension for Visual Studio, were used as well. The Git system and the GitLab code repository management system were used to control the project version.

This implementation of the web service was built on the basis of existing analogues. All their shortcomings were taken into account, in particular, the best security system of personal identification was developed, as well as innovative solutions for displaying search results on the map were added.

Keywords: Home rental search service, security, BankID.

Explanatory note size - 60 pages, contains 30 illustrations, 5 applications.

# ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

**до дипломної роботи**
**освітньо-кваліфікаційного рівня__бакалавр___**

на тему: “Веб сервіс пошуку оренди житла з покращеною системою ідентифікації особистості”

Київ – 2020 року

# ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

| № з/п | Формат | Позначення | Найменування | Кількість листів | Примітка |
|---|---|---|---|---|---|
| 1 | А4 | | Завдання на дипломний проект | 3 | |
| 2 | А4 | ІАЛЦ.467800.001 ВП | Відомість дипломного проекту | 1 | |
| 3 | А4 | ІАЛЦ.467800.002 ТЗ | Технічне завдання | 3 | |
| 4 | А4 | ІАЛЦ.467800.003 ПЗ | Пояснювальна записка | 60 | |
| 5 | А3 | ІАЛЦ.467800.004 Д1 | Схема бази даних | 1 | |
| 6 | А3 | ІАЛЦ.467800.005 Д2 | Діаграма класів контексту бази даних | 1 | |
| 7 | А3 | ІАЛЦ.467800.006 Д3 | Принципова схема пошуку житлових приміщень | 1 | |
| 8 | А4 | ІАЛЦ.467800.007 Д4 | Функціональна схема авторизації | 1 | |
| 9 | А4 | ІАЛЦ.467800.008 Д5 | Текст програми | 27 | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | *ІАЛЦ.467800.001 ВП* | | |
| *Змн.* | *Арк.* | *№ докум.* | *Підпис* | *Дата* | | | |
| *Розроб.* | | *Федорович В.Р.* | | | *Відомість дипломного проекту* | *Літ.* | *Арк.* | *Аркушів* |
| *Перевір.* | | *Роковий О.П.* | | | | | *1* | *1* |
| | | | | | | *НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІП-62* | | |
| *Н. Контр.* | | *Сімоненко В.П.* | | | | | | |
| *Затверд.* | | | | | | | | |

# ТЕХНІЧНЕ ЗАВДАННЯ

**до дипломної роботи**
**освітньо-кваліфікаційного рівня___бакалавр___**

на тему: “Веб сервіс пошуку оренди житла з покращеною системою ідентифікації особистості”

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ ..... 2

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ ..... 2

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ ..... 2

4. ДЖЕРЕЛА РОЗРОБКИ ..... 2

5. ТЕХНІЧНІ ВИМОГИ ..... 2

5.1. Вимоги до розробляємого продукту ..... 2

5.2. Вимоги до програмного забезпечення ..... 3

6. ЕТАПИ РОЗРОБКИ ..... 3

# 1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку програмного продукту по темі «Веб сервіс пошуку оренди житла з покращеною системою ідентифікації особистості». Область застосування: безпечний пошук житла для оренди.

# 2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

# 3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка веб-сервісу пошуку оренди житла з покращеною системою ідентифікації особистості.

# 4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, публікації в Інтернеті з даних питань.

# 5. ТЕХНІЧНІ ВИМОГИ

## 5.1. ВИМОГИ ДО РОЗРОБЛЯЄМОГО ПРОДУКТУ

- Зручний програмний інтерфейс.
- Можливість пошуку та публікації житлових приміщень для можливості їх оренди. Відображення таких приміщень на карті у вигляді маркерів.
- Безпечність використання веб-сервісу, що включає в себе можливість

авторизації за допомогою BankID та способи комунікацій в межах сервісу.

- Технічна коректність виконання усіх заданих алгоритмів.

### 5.2. ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Доступ до мережі Інтернет.

## 6. ЕТАПИ РОЗРОБКИ

| | Дата |
|---|---|
| Вивчення веб-сервісів аналогів, аналіз їх переваг та недоліків | 15.04.2020 |
| Складання і узгодження технічного завдання | 21.04.2020 |
| Створення модулів веб-сервісу, що розробляється | 24.04.2020 |
| Тестування окремих модулів системи | 21.05.2020 |
| Допрацювання, налагодження і виправлення помилок | 26.05.2020 |
| Оформлення документації дипломної роботи | 31.05.2020 |

# Пояснювальна записка
## до дипломного проекту

на тему: «Веб сервіс пошуку оренди житла
з покращеною системою ідентифікації особистості»

Київ – 2020

ЗМІСТ

ВСТУП ..... 3

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ..... 4

1.1. Визначення веб-сервісів пошуку оренди житла ..... 4

1.2. Аналіз існуючих веб-сервісів пошуку оренди житла ..... 4

1.2.1. Українські веб-сервіси пошуку оренди житла ..... 4

1.2.2. Іноземні веб-сервіси пошуку оренди житла ..... 7

1.3. Недоліки існуючих веб-сервісів пошуку оренди житла ..... 10

1.4. Концепція веб-сервісу на основі BankID ..... 11

ВИСНОВКИ ДО РОЗДІЛУ ..... 13

РОЗДІЛ 2. ВИЗНАЧЕННЯ ВИМОГ ТА ЗАДАЧ ПРОЕКТУВАННЯ ..... 14

2.1. Функціональні задачі веб-сервісів пошуку оренди житла ..... 14

2.2. Авторизаційна система веб-сервісів пошуку оренди житла ..... 16

2.3. Стек технологій для реалізації веб-сервісу пошуку оренди житла .. 18

ВИСНОВКИ ДО РОЗДІЛУ ..... 21

РОЗДІЛ 3. ВЕБ-СЕРВІС ПОШУКУ ОРЕНДИ ЖИТЛА З АВТОРИЗАЦІЙНОЮ СИСТЕМОЮ BANKID ..... 22

3.1. Опис запропонованого сервісу ..... 22

3.1.1. Опис реалізації основних функцій ..... 23

3.2. Авторизації запропонованого сервісу ..... 28

ВИСНОВКИ ДО РОЗДІЛУ ..... 30

РОЗДІЛ 4. ПРОГРАМНЕ РІШЕННЯ ТА ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ ..... 31
4.1. Опис програмного рішення системи ..... 31
4.3. Проектування бази даних ..... 32
4.2. Розробка серверної частини веб-сервісу ..... 34
4.2.1. Опис головних запитів до API ..... 34
4.2.2. Документація API проекту ..... 36
4.2.3. Опис взаємодії серверної частини сервісу з базою даних ..... 38
4.2.4. Модульне тестування коду ..... 39
4.2.4. Опис запитів в реальному часі ..... 41
4.3. Клієнтська частина додатку ..... 43
4.3.1. Реалізація запитів до API ..... 43
4.3.2. Опис нанесення маркерів на карту ..... 44
4.3.3. Оптимізація клієнтської частини додатку ..... 44
4.4. Технічна архітектура системи BankID ..... 45
4.4.1. Опис принципу авторизації ..... 45
4.4.2. Процедура авторизації BankID ..... 46
4.4.3. Збереження даних користувача в сервісі ..... 49
4.5. Система контролю версіями ..... 50
4.6. Інструкція користувача ..... 51
ВИСНОВКИ ДО РОЗДІЛУ ..... 56
ЗАГАЛЬНІ ВИСНОВКИ ..... 57
ПЕРЕЛІК ПОСИЛАНЬ ..... 59

# ВСТУП

**Актуальність.** Велика частина населення нашої країни користується послугами оренди житла. Це підтверджується великою кількістю оголошень на сайтах пошуку оренди житла та тим, наскільки швидко дані оголошення стають не актуальними в зв’язку зі здачею в оренду житла тій чи іншій людині. Не мало важливим є безпечність такої операції. На жаль, існують веб-сервіси оренди житла, користуючись якими люди займаються шахрайством, публікують не їхні або ж взагалі не справжні квартири.

Існують різні практики захисту від недостовірної інформації про квартири та будинки, такі як перевірка житла працівниками веб-сервісу з виїздом по адресі самого житла, перевірка документів на власність житла, перевірка за допомогою надсилання листа за вказаною адресою. Проте всі вони є або не достатньо безпечними, або не зовсім зручними.

Завданням цієї дипломної роботи є вирішення проблеми безпеки шляхом ідентифікації користувачів зручним для них способом. А також потрібно спроектувати систему, яка надавала б функціонал пошуку оренди житла та укладення договору його оренди безпосередньо в самому веб-сервісі.

**Об'єкт дослідження.** Веб-сервіс пошуку оренди житла.

**Предмет дослідження.** Безпечність процесу оренди житла.

**Наукова новизна.** Розроблений веб-сервіс дозволяє проводити усі дії пов’язані з орендою житла не залишаючи сайт, а також використовує систему авторизації, яка встановлює особистість користувача, що забезпечує надійний рівень захисту від шахрайства. Дана система не реалізована ні на одному українському, а також іноземному аналогічному веб-сервісі.

**Практична цінність.** Веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості дозволяє зручно та безпечно шукати житло для оренди та укладати договір оренди з його власником.

# РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

## 1.1. Визначення веб-сервісів пошуку оренди житла

Веб-сервіс пошуку оренди житла – це комплекс програмних рішень, які допомагають користувачам знаходити житло для оренди на визначений термін.

Різного роду сайти та мобільні додатки надають подібний функціонал у вигляді списку доступних квартир, будинків, які можна зручно відфільтрувати, зв’язатись з власником житла.

Існує декілька критеріїв оцінки таких сервісів. Найважливішим з них є критерій достовірності інформації наданої як орендатором житла так і власником житла. Мова йдеться про ідентифікацію осіб, між якими заключається договір оренди. Адже при недобросовісності однієї з сторін важко вирішити проблемні питання такі як псування майна, терміни оренди та виселення.

В даній роботі розглянуться альтернативні програмні рішення для пошуку оренди житла, які будуть використовувати вже існуючі практики та вдосконалювати їх функціонал, зокрема авторизацію та спосіб ідентифікації користувача для можливості реалізації ряду функціональних рішень для вирішення проблемних питань.

## 1.2. Аналіз існуючих веб-сервісів пошуку оренди житла

Розглянемо перелік існуючих веб-сервісів, які мають достатній функціонал для визначення веб-сервісу пошуку оренди житла. Проаналізуємо критерій безпеки та ідентифікації особистостей для кожного з них.

### 1.2.1. Українські веб-сервіси пошуку оренди житла

Серед українських можна виділити такі сервіси як:

- «100realty» (https://100realty.ua/)

- «dom RIA» (https://dom.ria.com/)
- «Лун» (https://lun.ua/)

Після реєстрації на сайті «100realty», є можливість з легкістю опублікувати об’єкт для оренди, власником якого являється інша людина. Ніякої перевірки на справжність житла та його власність немає. Також даний сайт не надає ніякої альтернативної можливості перевірки, як це прослідковується в описаному нижче сервісі «dom RIA».

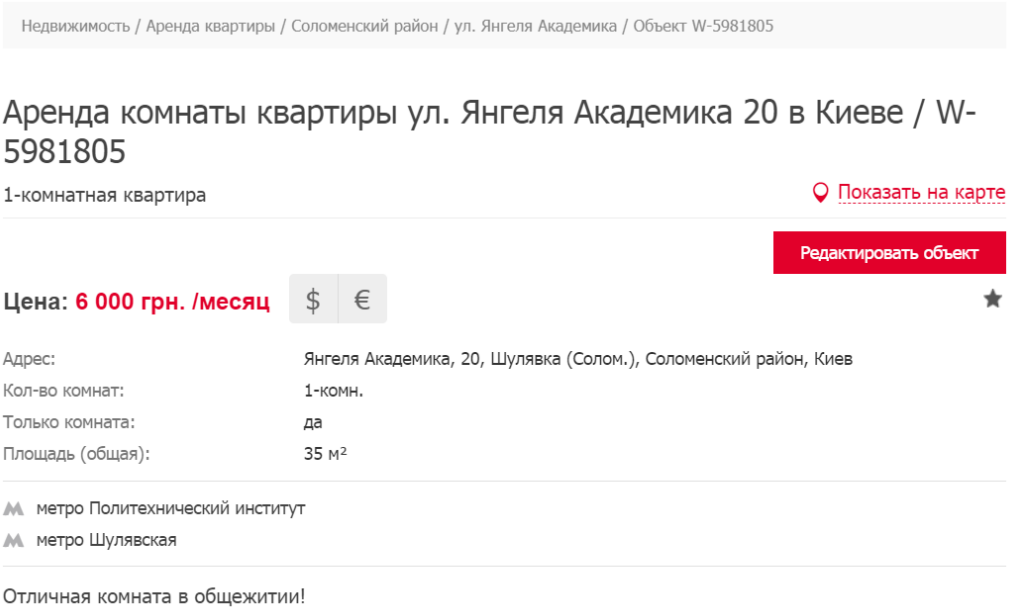



*Рисунок 1.1. Оголошення оренди житла не справжнього власника з сайту «realty100»*

Також даний сервіс не надає ніякого функціоналу для комунікації між сторонами договору оренди та узгодження деталей самої оренди, що змушує користувачів веб-сервісу обговорювати деталі оренди поза сервісом. Разом з тим немає способу моніторингу квартир та домів, які орендуються, тобто відсутня гарантія зйому квартири лише однією людиною. Тому орендатор не може бути впевненим у тому, що після узгодження огляду житла чи його оренди, воно не буде орендоване іншою особою.

Перевіривши наступний сайт – «dom RIA», на безпеку та ідентифікацію особистості, опублікувати оголошення виявилось не складніше, ніж на

«realty100», проте даний сайт відрізняється альтернативним способом підтвердження власності житла та його справжньості.

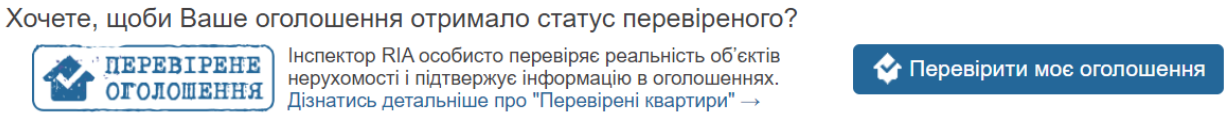



*Рисунок 1.2. Альтернативний спосіб підтвердження власності та справжньості житла на сайті «dom RIA»*

Процедура перевірки заключається у виклику працівника компанії «dom RIA». Після залишення заявки на перевірку оголошення, вказується дата візиту. Під час огляду житла працівник оглядає його та фотографує на спеціальну панорамну камеру 360 градусів. Також працівник звіряє усю інформацію, вказану в оголошенні, включаючи тип здачі (від власника чи посередника). Після успішної перевірки житла на сайті з'являється відмітка про перевіреність житла.

Комунікація, між користувачами веб-сервісу відбувається за його межами, по аналогії до «dom RIA». Функціонал моніторингу також відсутній.

Переглянувши наступний український веб-сервіс для оренди житла – «Лун», можна зробити висновок, що на даному сайті відсутня реєстрація. Самі ж оголошення про оренду житла сервіс збирає з інших сайтів, включаючи «realty100» та «dom RIA». Будь-якої перевірки на оренду житла даний веб-сервіс не надає. Тому кожен має можливість опублікувати оголошення на сайті з недостатньою перевіркою на справжність житла та бути опублікованим на сервісі «Лун» також.

Щодо ідентифікації з боку орендатора житла, то кожен не зареєстрований користувач одного з даних сайтів може з легкістю побачити номер власника житла або посередника та зв'язатись з ним, що також доволі небезпечним або не зовсім зручним. Так як при здачі житла, маючи лише контактні дані орендатора, власнику потрібно буде відсканувати або

сфотографувати його документи та підписати договір для того, щоб бути переконаним в тому, що орендар буде дотримуватись умов оренди.

В зв'язку з відсутністю реєстрації і ведення аккаунту користувача, функціонал комунікації між користувачами в межах сервісу відстутній. Моніторинг орендованого житла також не реалізований.

### 1.2.2. Іноземні веб-сервіси пошуку оренди житла

Серед іноземних можна виділити такі сервіси як:

- «Booking» https://www.booking.com/
- «Airbnb» https://www.airbnb.com/

Сервіс «Booking» не надає можливість опубліковувати оголошення без перевірки на власність житла. Процедура перевірки відбувається способом листування на звичайну пошту. Для того, щоб оголошення було опубліковане, користувачу доведеться вказати справжню адресу житла для оренди та отримати лист з активаційним кодом, після чого ввести його на сайті сервісу.

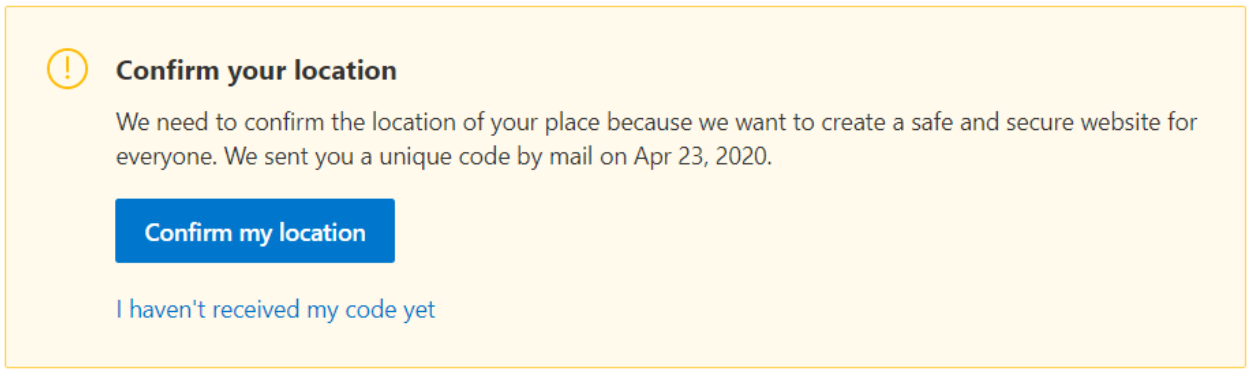



*Рисунок 1.3. Сповіщення про перевірку житла на сайті «Booking»*

На відміну від українських аналогів, даний сервіс надає можливість моніторингу житла. Усі орендовані квартири, будинки, номери готелів записуються в історію аккаунту. Є можливість перегляду як усього орендованого житла за час перебування на сервісі, так і відмінених бронювань. Хоч це пояснюється більшим акцентом на подобову оренду житла, даний функціонал був би зручним та забезпечив би додатковий рівень безпеки і для

довготривалої оренди.

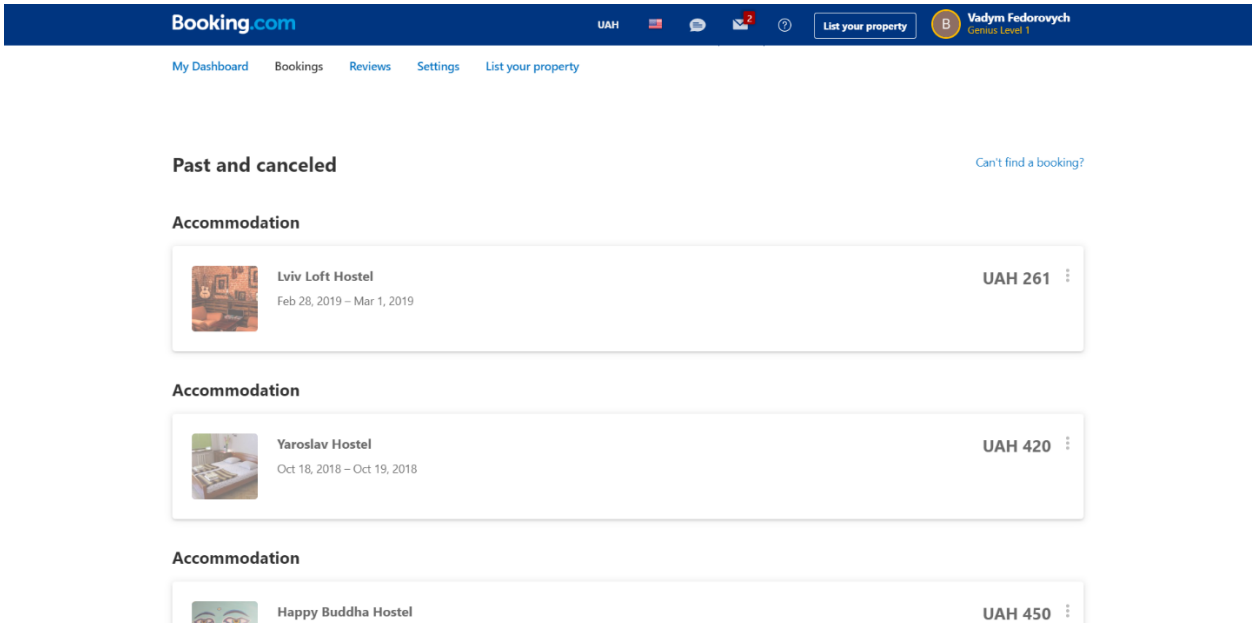



*Рисунок 1.4. Історія бронювань житла*

Також даний сервіс дає змогу листування за допомогою месенджера, перейти на який є можливість за допомогою іконки «повідомлення». Даний месенджер є зручним у використанні та надає змогу комунікувати між орендатором та власником житла безпосередньо в межах сервісу.

Розглянемо іноземний аналог веб-сервісу пошуку оренди житла «Airbnb». Щоб опублікувати оголошення на «Airbnb», потрібно додати фото ID-картки (паспорту, водійських прав), перевірити номер телефону та вказати свою кредитну карту.

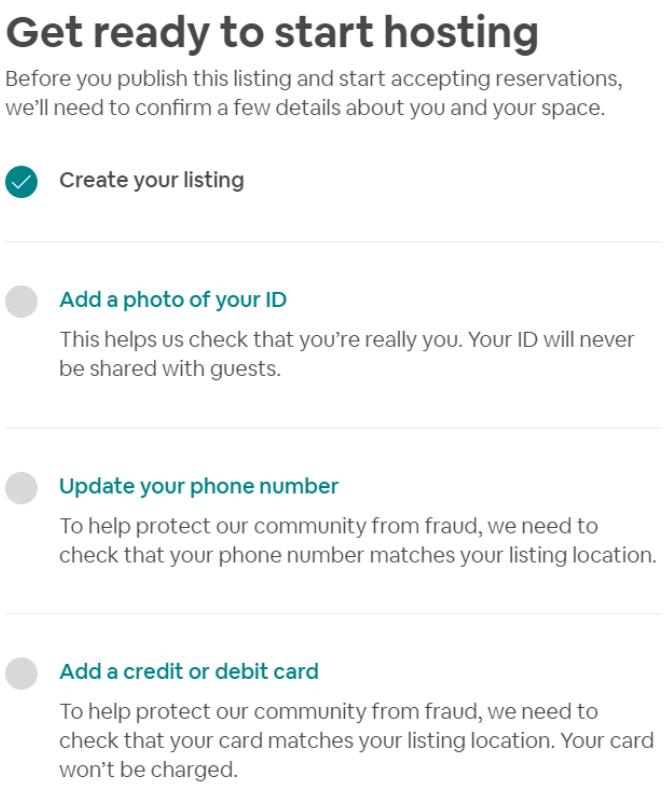



*Рисунок 1.5. Критерії розміщення оголошення на «Airbnb»*

Також «Airbnb» в деяких випадках гарантує компенсацію в розмірі $1,000,000 для власника житла.

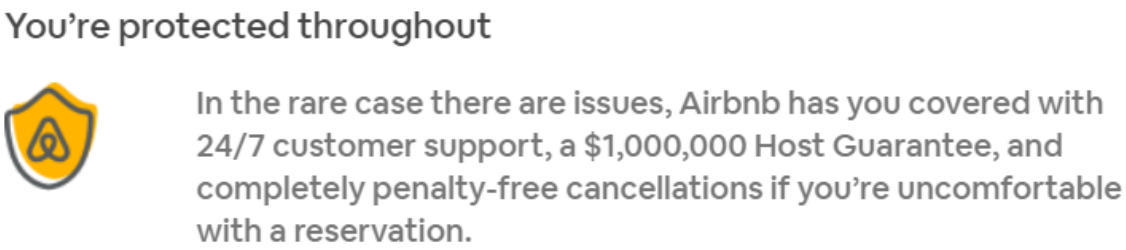



*Рисунок 1.6. Гарантія від «Airbnb»*

Під час реєстрації заявки на публікацію оголошення про оренду житла, було вказано найбільше з всіх попередніх сервісів інформації про житло та про його власника. Більшість з полів інформації були обов'язковими, включаючи ID-картку, номер телефону та банківську картку власника житла.

Можливість листуватиння у месенджері сервісу також наявна. Моніторинг відбувається по всіх орендованих квартир, будинків, номерів

готелю та тих, які плануються орендуватись.

На відміну від українських аналогів «Booking» та «Airbnb» не дозволяє резервувати або орендувати житло не авторизованим користувачам, що значно покращує рівень захисту ідентифікацій особистостей та рівень довіри до орендаторів.

### 1.3. Недоліки існуючих веб-сервісів пошуку оренди житла

Проаналізувавши 5 різних сервісів пошуку оренди житла, можна зробити висновок по кожному з них.

Сайт «100realty» виявився найменш безпечним щодо ідентифікації сторін договору оренди. На даному сайті можна з легкістю опублікувати будь-яке оголошення вказавши лише номер телефону, який підтверджується за допомогою SMS-повідомлення. Комунікація між сторонами договору оренди відбувається поза сервісом, що робить її менш безпечною. Відсутній моніторинг статусу орендованого житла.

Веб-сервіс «dom RIA» має аналогічний функціонал як і попередньо описаний сайт, за виключенням опції перевірки житла працівником компанії, яка є необов'язковою, тому даний сайт також можна вважати небезпечним щодо ідентифікації особистостей.

Сервіс «Лун» лише збирає з усіх інших сайтів інформацію, тому захист інформації покладається на інші сайти, з яких були зібрані оголошення. Даний принцип також не є безпечним, адже інформація може бути зібрана з таких же сервісів як «100realty» та «dom RIA», які не мають достатнього захисту ідентифікації особистостей.

Аналізуючи іноземні сайти, можна прийти до висновку, що критерій захисту ідентифікації особистостей має більшу значимість.

«Booking» має доволі не складний, проте ефективніший, ніж українські аналоги спосіб ідентифікації через листування. Недоліком залишається лиш те, що поштові ящики не завжди є приватними, тобто можуть не мати замка

під ключ і бути в загальному доступі.

«Airbnb» має практично ідеальний спосіб ідентифікації особистей, який полягає в наданні, такої інформації як ID-картка та банківська карта. Недоліком являється надто складна реєстрація оголошення. Для того, щоб опублікувати оголошення, власнику потрібно відразу вказати всі дані про житло, прикріпити фотографії, включаючи фото ID-картки, яка не завжди може бути під рукою.

### 1.4. Концепція веб-сервісу на основі BankID

BankID – спосіб підтвердження особистості громадян за допомогою українських банків для надання адміністративних та інших послуг онлайн.

Розглянемо принцип роботи BankID:

1. Користувач обирає спосіб ідентифікації «Увійти через BankID».
2. На сторінці, куди користувач був переадресований, обирає свій банк.
3. Користувач переглядає інформацію, яка про нього буде надіслана веб-сервісу та підтверджує згоду авторизувавшись за допомогою інтернет-банку, який він використовує.
4. Проходить двофакторну аутентифікацію, вводячи SMS-пароль (даний етап може відрізнятись в залежності від банку).
5. Уся інформація, яку запросив веб-сервіс, передається йому. [1]

Дана концепція дозволяє з легкістю авторизуватись користувачу на будь-який сайт, зокрема веб-сервіс пошуку оренди житла, та надати усю потрібну інформацію, в разі виникнення проблемних ситуацій.

З рівнем безпеки такий принцип можна порівняти з реєстрацією оголошення на сайті «Airbnb». Проте BankID вирішує проблему ідентифікації набагато зручніше. Зникає потреба фотографувати або копіювати свої документи, після чого надсилати їх стороннім сайтам. BankID автоматично надсилає всю необхідну інформацію та контролює її конфіденційність.

Для достатньої захищеності щодо ідентифікації особистостей веб-сервісу буде достатньо лише копії паспорту як ідентифікатора. Не потрібно буде дізнаватись повністю всю інформацію про користувача, адже концепція система безпеки передбачає передачу ідентифікатора лише у таких випадках як шахрайство. Завдяки цьому правоохоронні органи з легкістю зможуть віднайти порушника та вирішити усі проблемні питання.

# ВИСНОВКИ ДО РОЗДІЛУ

Незважаючи на існування великої кількості веб-сервісів для пошуку оренди житла, усі вони мають недоліки в системі безпеки або в зручності її використання. Існуючі на українському ринку сервіси мають недостатній рівень безпеки. Лише деякі іноземні концепції ідентифікації особистостей були достатньо захищеними для їх використання, проте мали проблеми у незручності використання.

Для покращення практик усіх проаналізованих сайтів необхідно обрати систему, яка була б зручною у використанні, а також мала достатній рівень захищеності. Такою системою є BankID, концепція якого була розглянута.

# РОЗДІЛ 2. ВИЗНАЧЕННЯ ВИМОГ ТА ЗАДАЧ ПРОЕКТУВАННЯ

## 2.1. Функціональні задачі веб-сервісів пошуку оренди житла

Веб-сервіси пошуку оренди житла призначені для перегляду доступних до зйому квартир або будинків з можливістю подальшої комунікації з власником житла. Для останнього це сервіс для публікації оголошень оренди його житла. Решта функцій, за виключенням авторизації, (рейтинг користувачів та житла, відгуки про житло, орендатора або власника житла) є додатковим функціоналом.

Задля забезпечення потрібного функціоналу, користувачі веб-сервісу повинні бути авторизованими. Саме задачі авторизації відведена вагома частина даної дипломної роботи, адже вона є засобом для ідентифікації користувача, який слугує основним елементом безпеки в процесі пошуку та оренди житла.

Розглянемо основні функціональні задачі веб-сервісу пошуку оренди житла більш детально. Серед них можна виділити такі задачі як:

- Публікація оголошення про можливість оренди житла
- Перегляд доступних для оренди квартир та будинків
- Комунікація між орендатором та власником житла
- Моніторинг орендованого житла

Задача **публікації оголошення про можливість оренди житла** полягає в заповненні інформації про квартиру або будинок, який власник житла планує здати в оренду та публікації такої інформації в списку усіх доступних житлових приміщень. Щоб оголошення було інформативним, потрібно вказати достатньо інформації про нього, щоб орендатор зміг собі відповісти на усі цікаві йому питання.

Серед інформації, яка повинна бути розміщена, можна виділити:

- Фотографії усіх кімнат, включаючи кухню, балкон (якщо він присутній)
- Адреса, за якою знаходиться житло

- Короткий опис житла, який має включати усі деталі, які могли б вплинути на прийняття рішення орендатора
- Актуальність пропозиції оренди житла
- Ціна оренди на визначений період часу

Дана інформація має бути актуальною та відповідати дійсності. Для цього потрібно надати можливість редагувати її, оновлюючи усі дані та актуальність пропозиції взагалі.

Якщо пропозиція оренди житла не є актуальною, потрібно надати користувачу можливість її видалення. Також буде корисним зробити фільтр актуальності житла в пошуковому запиті. Адже інформацію, яка не оновлялась більше тижня можна вважати не актуальною.

**Перегляд доступних для оренди квартир та будинків** потребує відповідного відображення на сайті веб-сервісу та можливостей фільтрованого пошуку потрібного житла. Порівнюючи та аналізуючи різні реалізації подібних сервісів, які були розглянуті у попередньому розділі, можна виділити такі фільтри пошукових запитів:

- Тип житла (квартира / будинок)
- Кількість кімнат
- Ціна за квартиру
- Дата публікації та її актуальність
- Поверх (якщо це квартира)
- Район населеного пункту
- Термін здачі (подобово, довготривала оренда)
- Тип здачі (від власника житла, або ж від посередника)

Дані фільтри є ключовими в процесі пошуку житла для оренди. Додатковими фільтрами можуть бути такі фільтри як рік будівництва, тип будинку, стіни, висота стелі, опалення, площа житла.

**Комунікація між орендатором та власником житла** повинна бути

реалізована зручним способом. У всіх українських аналогів даного веб-сервісу комунікація між сторонами договору оренди реалізована лише у вигляді номеру телефону власника житла на сторінці оголошення, далі вся комунікація відбувається поза межами веб-сервісу, що є не зовсім безпечним способом. В розглянутих іноземних сервісах, комунікація відбувається безпосередньо через сам веб-сервіс. Що є безпечнішим, та у разі винекнення проблемних ситуацій буде вирішуватись за допомогою даних (інформації про оренду), які сторони передають до веб-сервісу.

Тому дана задача потребує інструменту комунікації, за допомогою якого користувачі вирішували б усі питання в межах сервісу.

Для того, щоб користувачі могли вести **моніторинг орендованих квартир**, як зі сторони власника житла, так і зі сторони орендатора, веб-сервіс пошуку оренди житла повинен надавати функціонал, який задовільнив би дану потребу. Для цього достатньо реалізувати сторінку з списком житлових приміщень, кожне з яких має таку інформацію як контакт орендатора / власника житла та статус оренди, який відповідає дійсності. Також при розірванні договору оренди потрібно видаляти оголошення про квартиру / будинок з моніторингу оренди та записувати їх в історію оренд.

Решта функцій, таких як **рейтингова система користувачів**, **можливість залишати відгуки** про житло, орендаторів та власників житла буде потребувати відповідної релевантної системи відображення інформації. Потрібно реалізувати таку систему, яка б фільтрувала фейкові відгуки та оцінки. Дана задача потребуватиме аналізу інформації про користувача на сервісі, такої як тривалість перебування користувача в сервісі, кількість оренд та здач в оренду житла.

### 2.2. Авторизаційна система веб-сервісів пошуку оренди житла

Функціональну задачу авторизації на веб-сервісі розглянемо більш детально та визначимо критерії, які потрібно враховувати при проектуванні

авторизаційної системи.

Проаналізувавши системи авторизацій та ідентифікацій особистостей на сайтах аналогах, потрібно поставити задачу реалізації таким чином, щоб уникнути всіх недоліків розглянутих у попередньому розділі. Потрібно щоб при реєстрації користувач не вказував надто багато інформації, для зручності використання, проте, щоб цієї інформації було достатньо для його ідентифікації. Використання систем авторизації по типу BankID задовільнила б ці потреби.

Серед даних, які можуть бути передані для ідентифікації особистості BankID пропонує нам наступну інформацію про:

- ПІБ
- Стать
- Дата народження
- Ідентифікаційний номер
- Адреса реєстрації
- Контактний номер телефону
- Електронна пошта
- Копії паспорту та ідентифікаційного коду [2]

Для ідентифікації особистості достатньо буде лише одного з двох унікальних та незмінних даних, якими є ідентифікаційний номер та скан-копії паспорту та ідентифікаційного коду. Цього буде достатньо для ідентифікації особистості та впевненості в тому, що людина має єдиний аккаунт.

Щодо зручності використання, BankID дозволяє авторизуватись через 14 українських банків, тому категорія людей, яка зможе скористатись такою авторизацією є достатньо великою.


*Рисунок 2.1. Перелік банків, які використовує авторизаційна система BankID*

2.3. Стек технологій для реалізації веб-сервісу пошуку оренди житла

Враховуючи те, що проект сервісу пошуку оренди житла є доволі великомасштабним, важливо вибрати мову програмування, яка дозволила б зпроектувати таку систему.

Розглянемо популярні варіанти мов програмування для реалізації веб-сервісів. Такими мовами є:

- Java (Фреймворк Spring)
- C# (Фреймворк .NET Core)
- PHP (Фреймворк Laravel)
- Python (Фреймворк Django)
- JavaScript (Платформа Node.js)

Одразу відкинемо скриптові мови програмування, тому що за допомогою них важко підтримувати код великого проекту. Дані мови слугують для сайтів та веб-сервісів меншого масштабу, такими мовами являються PHP, Python та JavaScript. Не дивлячись на те, що JavaScript має велику підтримку з боку платформи Node.js, все ж дана мова програмування залишається скриптовою та важко підтримуванною у глобальних проектах.

Залишається два популярних варіанта мов програмування, таких як C#

та Java. Обидві мови програмування є зручні у випадку розширення проекту та його підтримки, надають великий функціонал. Існує багато фреймворків, створених саме для цих мов програмування, які дозволять швидко реалізовувати поставлені задачі. Проте C# як мова програмування є прогресивнішою. Її розвиток як мови, так і фреймворків написаних для неї є значно швидшим, з цього випливають як плюси, так і мінуси.

Мінуси C# в тому, що літератури для Java значно більше, ніж для C#, в силу того, що Java більш стабільніша і менше змінюється. Щодо C#, не так багато літератури можна знайти для сучасних версій фреймворка, який використовується як головний у багатьох проектах, .NET Core. Проте цей мінус не є значним, тому що в документації Microsoft можна знайти все що потрібно.

Плюси C# та .NET Core в більшій функціональності та в кращій її оптимізованості. Хорошим прикладом та достатнім аргументом, щоб обрати C# слугує новий фреймворк Blazor WebAssembly. Якщо в Java ми зможемо написати лише серверну частину проекту, а для візуалізації даних на клієнтській частині ми змушені будемо використовувати JavaScript або мови, які в нього компілюються, то в сучасному C# / .NET Core використовуючи Blazor WebAssembly ми маємо змогу писати клієнтську частину безпосередньо на C#. Даний фреймворк не компілює C# код в JavaScript та не генерує html код на серверній частині, а виконує запрограмовану логіку саме на клієнтській частині. Все це можливо завдяки сучасній технології WebAssembly.

Так як я один розробляю як клієнтську частину сайту, так і серверну, простіше використовувати одну мову в обох випадках, ніж функціонал декількох.

Після того як ми обрали мову програмування та основний фреймворк .NET Core, розглянемо детальніше, які технології та фреймворки будуть використані при написанні веб-сервісу.

Веб-сервіс буде реалізований у вигляді сайту, проте в перспективі для більшої зручності реалізації також потребуватиме частина користувачів, які активно використовують смартфони та мобільні додатки. Тому потрібно створити API проект, який би слугував для надання інформації як для сайту, так і для мобільних додатків на платформі Android та iOS. Для цього використаємо фреймворк ASP.NET Core, та його варіацію проекта API.

Для клієнтської частини, як було описано вище, буде обрано фреймворк Blazor WebAssembly. Він містить весь потрібний функціонал, який потрібен для написання проекту, заміняючи в повній мірі такі фреймворк як Angular та бібліотеку React та в більшій мірі заміняючи JavaScript. Останній буде використовуватись лише в візуальній частині проекту, реалізовуючи анімації та інші візуальні ефекти клієнтської частини, адже Blazor WebAssembly досі, на жаль, не має підтримки роботи з DOM елементами, та в силу своїх особливостей навряд чи буде її мати.

Що стосується бази даних, то переглядаючи можливі варіанти систем управління базами даних, такі як MS SQL, MySQL, PostgreSQL, було вибрано останню, так як PostgreSQL є найбільш функціональнішим і оптимізованішим. Фреймворком, який буде сполучати API частину проекту та базу даних, було обрано Entity Framework Core як найбільш функціональна та оптимізована технологія доступу до баз даних для C# / .NET Core.

Усі дані, які будуть потребуватимуть передачі в реальному часі, будуть передані за допомогою фреймворка SignalR2. Даний фреймворк забезпечить надійне сполучання та передачу даних по веб-сокету або ж звичайному запиту через long polling.

# ВИСНОВКИ ДО РОЗДІЛУ

Розглянуті усі функціональні задачі проекту такі як публікація оголошення про оренду житла, перегляд доступних для оренди житла та комунікація між орендатором та власником житла. Також проаналізований додатковий функціонал, який включає в себе рейтингову систему та систему відгуків про користувачів або житло. Детальніше розглянута система авторизації. Задані усі відповідні критерії для виконання поставлених задач.

Було обрано мову програмування та фреймворки, які будуть використані при розробці проекту, на основі розмірів та типу проекту, а також на основі функціоналу, який дані мови програмування та фрейворки представляють.

# РОЗДІЛ 3. ВЕБ-СЕРВІС ПОШУКУ ОРЕНДИ ЖИТЛА З АВТОРИЗАЦІЙНОЮ СИСТЕМОЮ BANKID

## 3.1. Опис запропонованого сервісу

Пропонується сервіс, який реалізовує весь достатній для зручного пошуку оренди житла та її здачі функціонал, та використовує систему авторизації BankID. Також даний сервіс надає весь функціонал, який потрібен для комунікації та узгодження усіх подробиць договору оренди, таких як месенджер та моніторинг квартир які здаються та квартир які орендуються.

Для сервісу передбачається три ролі, таких як:

- Орендатор
- Власник житла
- Адміністратор сервісу

Договір оренди заключається між орендатором та власником житла, адміністратор сайту в свою чергу виконує функції модератора, такі як допомога у вирішені питань користувачів та фільтрація контенту (інформацію про житло та його оренду), який додають користувачі сервісу.

В запропонованій системі власник житла матиме можливість опублікувати оголошення про доступне житло для оренди, переглядати свої квартири, які здаються, залишати відгук про орендатора та комунікувати з орендаторами його житла. Усі попередньо названі функції користувач в ролі власника житла зможе виконувати лише після того, як буде авторизованим.

Користувач в ролі орендатора, після того, як авторизується за допомогою системи BankID зможе використовувати наступний функціонал: переглянути всі житлові приміщення, які орендуються, залишати відгуки про власника житла та житло, комунікувати з власником житла, у якого він його орендує.

В неавторизованому вигляді користувач матиме змогу використовувати роль потенційного орендатора та переглядати всі квартири

та будинки доступні для оренди.


*Рисунок 3.1. Use-Case діаграма*

3.1.1. Опис реалізації основних функцій

Для реалізації функції пошуку оренди житла було вибрано 6 критеріїв, таких як: мінімальна та максимальна ціна, кількість кімнат, тип здачі, тип житла, поверх.

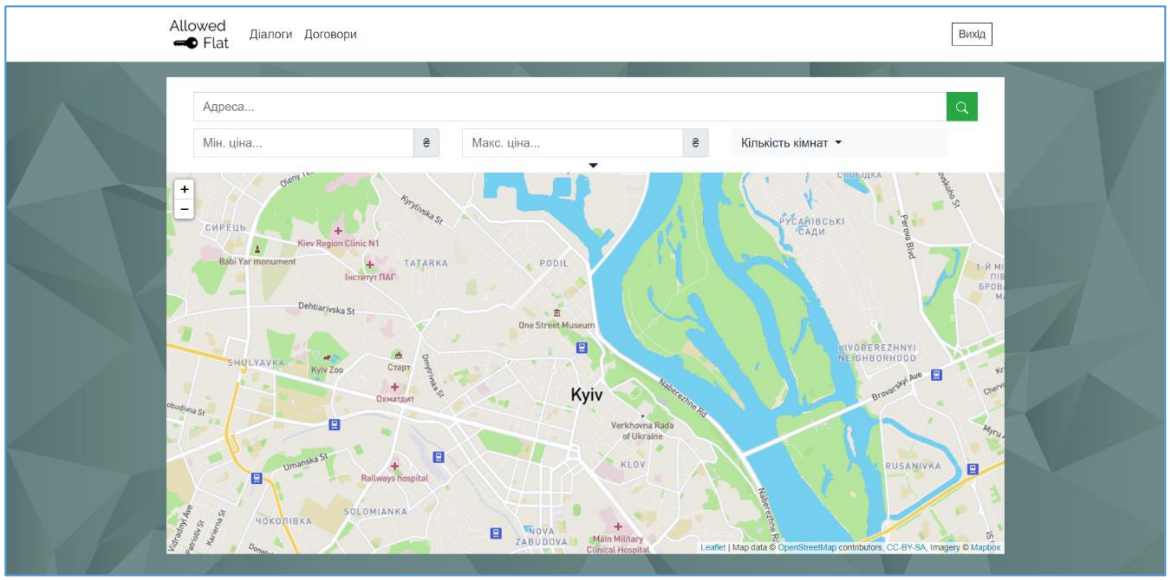



*Рисунок 3.2. Головна сторінка з фільтрами*

Такі фільтри як тип здачі, тип житла, поверх та сортування по вибраному критерію були винесені в окреме випадаюче меню, як фільтри, яким не надають значення в першу чергу.

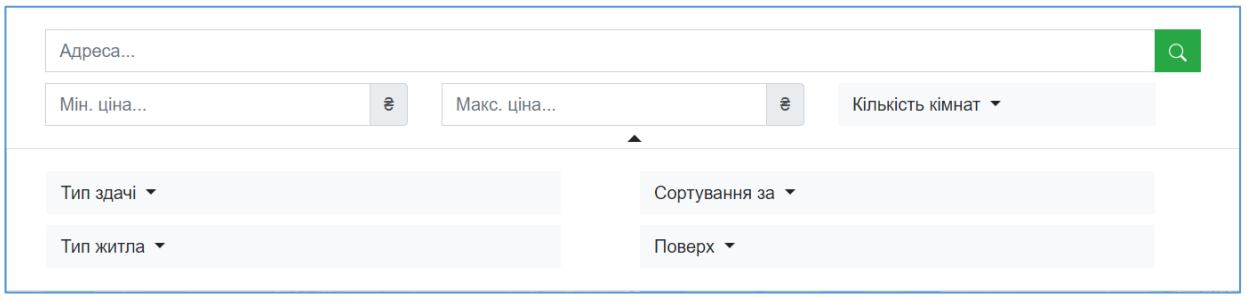



*Рисунок 3.3. Випадаюче меню з додатковими фільтрами*

Сортування відбувається за такими критеріями як: ціна (за спаданням та за зростанням), актуальністю (новіші та давніші оголошення). Та на відміну від веб-сервісів аналогів система сортування відображає результати на карті за допомогою кольорових маркерів. Найбільш релевантні пошукові результати позначаються зеленим маркером. Маркери результатів середньої релевантності мають жовтий колір. Найбільш не релевантні результати використовують маркери червоного кольору.

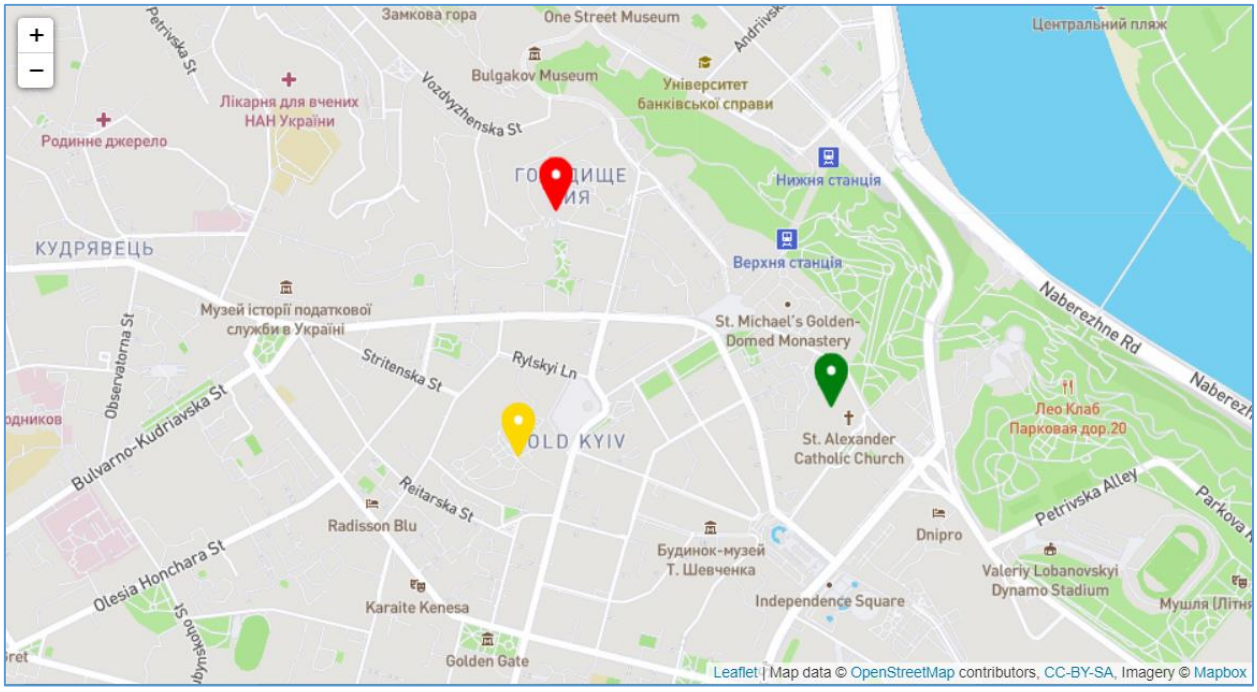



*Рисунок 3.4. Маркери сортування*

Якщо порівнювати з розглянутими раніше сайтами аналогами, то дана система відображення сортованих результатів є більш наглядною для користувача. Користувачу важливіше місце розташування житла, а лише після того поле, по якому йде сортування. В веб-сервісах аналогах система пошуку навпаки пріоритетним ставить критерій сортування.

Публікація оголошень, які потрапляють у результати на пошуковий запит в запропонованій системі відбувається наступним чином.

Користувач, який публікує оголошення повинен бути авторизованим через BankID для перевірки ідентифікації його особистості. Після чого для нього стане доступним пункт меню та відповідна сторінка, де він зможе вказати інформацію про свою квартиру / будинок та вказати дане житло на карті за допомогою маркера.

Інформація про житло має обов’язкові поля для заповнення, такі як:

- Адреса житла, яка включає в себе інформацію про місто, вулицю та будинок (хоч місцерозташування задається маркером, проте для користувачів буде доступний також пошук по назві вулиці / місту)
- Маркер, який вказує на розташування житла (даний маркер задає широту та довготу житла на карті, для подальшого пошуку по заданим критеріям)
- Ціна (даний пункт є обов’язковим, так як житло без вказаної ціни не є релевантним результатом пошуку, ціна – один з головних критеріїв пошуку житла)
- Кількість кімнат (аналогічно до ціни, кількість кімнат є одним з головних критеріїв пошуку житла)

Додаткові поля про житло, такі як тип здачі, тип житла (квартира чи будинок) та поверх є необов’язковими, та при загальному пошуковому запиті враховуватись не будуть.

Перегляд орендованих квартир / будинків, та ті які здаються буде реалізовано у вигляді списку усіх житлових приміщень з статусом здачі

(опубліновано, здається) та посиланням на бесіду в месенджері даного веб-сервісу. Такий список, навіть якщо він буде містити лише одну квартиру / будинок, буде слугувати зручним моніторингом за житлом, яке орендується та здається, а також буде надавати змогу одразу перейти в месенджер з орендатором / власником житла для вирішень будь-яких запитань.

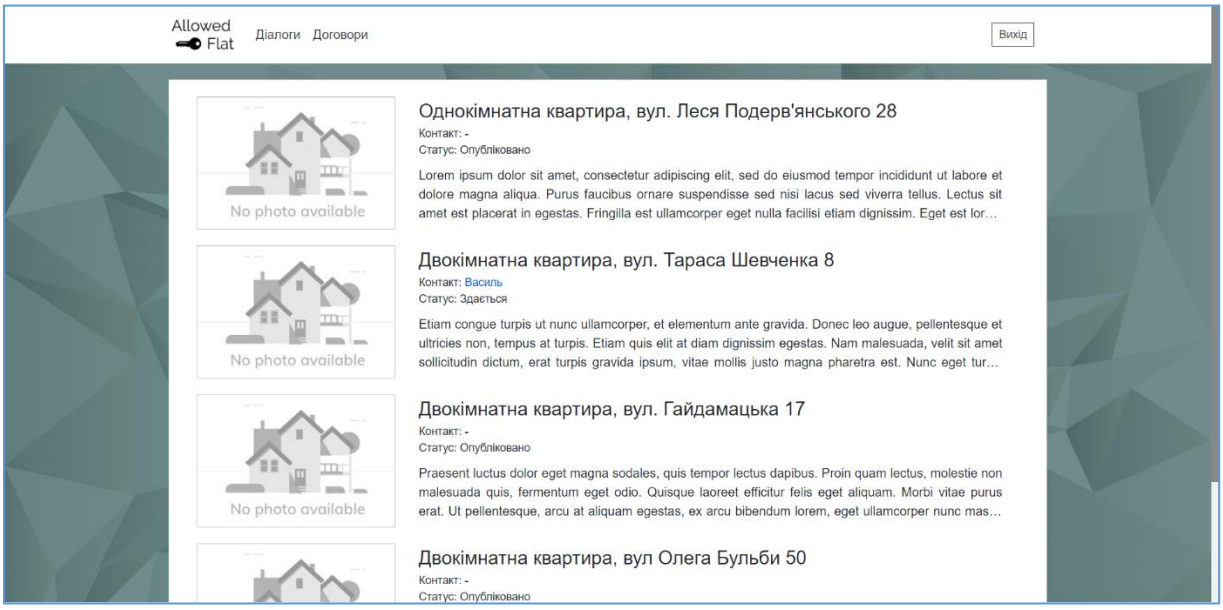



*Рисунок 3.5. Список житлових приміщень (квартир / будинків), які здаються*

Для того, щоб зв’язатись з орендатором / власником житла в запропонованій системі розроблений месенджер з мінімальним, проте забезпечуючи достатнім для комунікації, функціоналом. Запропонований месенджер створений для того, щоб користувачі обговорювали всі питання не виходячи за межі сервісу, адже лише в сервісі, користувач впевнений, що він комунікує саме з власником житла. Навіть якщо даний користувач не являється власником житла, за допомогою системи BankID легко буде ідентифікувати того чи іншого користувача, та у разі шахрайства передати інформацію правоохоронім органам.

Месенджер надає можливість користувачам зберігати діалоги з іншими користувачами веб-сервісу, надсилати їм повідомлення та переглядати вже надіслані, спостерігати за онлайн статусом співрозмовника.

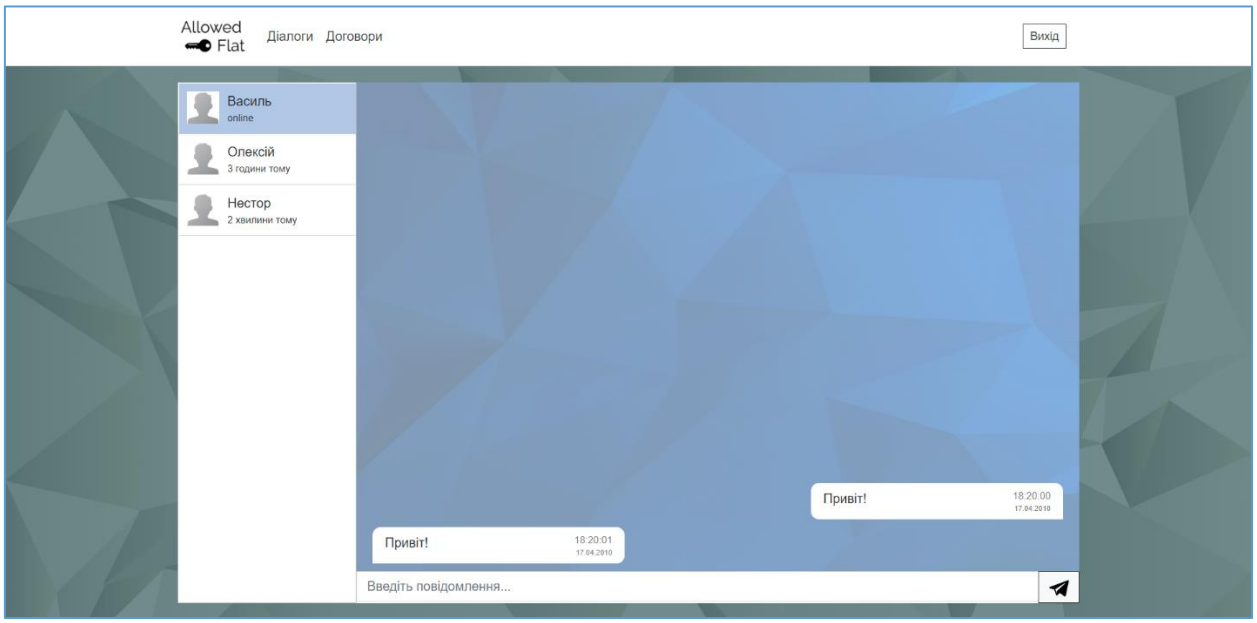



*Рисунок 3.6. Месенджер*

Система відгуків надає ще один рівень захисту до ідентифікацій особистостей та їх квартир / будинків. Так як з житлом все може бути впорядку в плані відповідності ціни, адреси та кількості кімнат, проте дана система не враховує реальний стан житла, відношення власника житла та орендатора стосовно оренди. Тому для запропонованого веб-сервісу була розроблена система оцінок та відгуків про житло, власника житла та орендатора.

Користувач, який орендує житло та користувач, який являється власником житла можуть залишити відгук про надані послуги. Таким чином даний критерій оцінки також буде використаний для релевантності житла або для оцінки власником житла орендатора.

Оцінка житла та власника житла буде враховуватись при виведенні на карті маркерів відповідного кольору, а відгук буде залишений під описом даної квартири / будинку. Власник житла не матиме змоги його видалити для того, щоб наступні орендатори отримували актуальні дані оголошення. Також при повторній публікації оголошення про оренду житла, відгук, який стосується конкретного житла, буде перенесений на оголошення за такою ж адресою.

Відносно користувача, який здає в оренду житло, при запиті на оренду, тобто першому повідомлені, яке буде йому надіслано, буде відображатись оцінка орендатора та відгуки про нього. Зрозуміло, що видалити дані відгуки буде неможливо, щоб власники житла оцінювали можливості здачі оренди житла тому чи іншому користувачу.

Зважаючи на те, що оцінки можуть бути суб'єктивними, при виведені середньої оцінки користувача або обрахунку релевантності результату запиту будуть враховуватись оцінки користувачів, які залишили відгук.

Також при винекнені конфліктних ситуацій, надається можливість написати адміністрації сайту для перегляду оцінок.

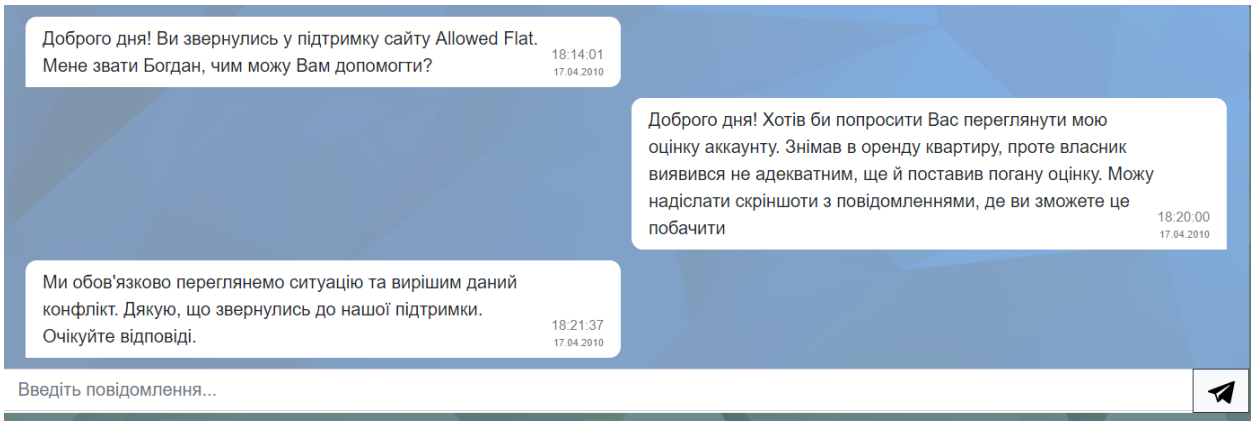



*Рисунок 3.7. Комунікація користувача з тех. Підтримкою сайту*

Всі сповіщення про перегляд оцінок, проставлення нових оцінок, зміну та видалення поставлених оцінок надсилаються користувачу на електронну адресу. Це реалізовано для того, щоб користувач зміг якнайшвидше дізнатись про оновлення свого рейтингу та вчасно зреагувати.

### 3.2. Авторизації запропонованого сервісу

Для авторизації використано розглянуту раніше систему BankID.

З боку користувача сторінка авторизації буде виглядати як кнопка входу, яка переадресовуватиме на сторінку вибору банку, після чого користувачу потрібно буде обрати банк, клієнтом якого він є. Після вибору банку, користувач буде перенаправлений на сторінку авторизації обраного

банку для процедури ідентифікації. Після успішного проходження процедури ідентифікації, користувач переадресовується на сторінку запропонованого веб-сервісу пошуку оренди житла. В той же час сервіс отримує зашифровану інформацію користувача від банку клієнта і авторизує клієнта вже на стороні самого веб-сервісу. [3]

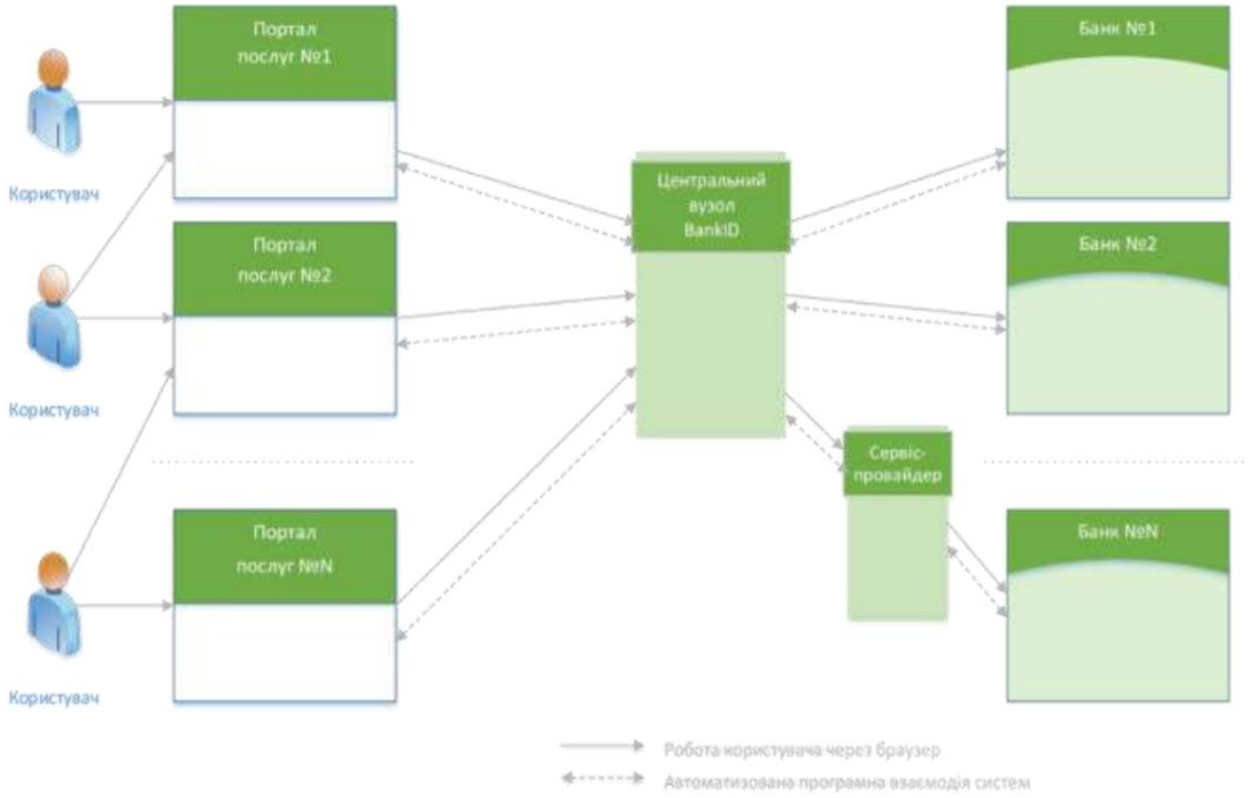



*Рисунок 3.8 Загальна схема функціонування системи BankID НБУ*

# ВИСНОВКИ ДО РОЗДІЛУ

Запропонований веб-сервіс, оснований на ідентифікації особистостій користувача за допомогою BankID та переліком нововведень відносно сервісів аналогів по типу сортування за кольором маркера, релевантність запитів на основі відгуків, може бути суттєвим конкурентом веб-сервісів пошуку житла як мінімум на українському ринку та надати можливість користучам сервісу безпечно орендувати та здавати житлові приміщення, комунікуючи і обговорюючи договор оренди житла не виходячи за межі веб-сервісу.

# РОЗДІЛ 4. ПРОГРАМНЕ РІШЕННЯ ТА ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ

## 4.1. Опис програмного рішення системи

Обравши мову програмування в розділі вище, оцінимо можливості та доступні інтерфейси веб-сервісу, їх зв'язок один між одним.

Для того, щоб з мінімальними витратами ресурсів досягти бажаного результату була вибрана технологія Blazor WebAssembly, яка була описана вище. Замість того, щоб створювати інтерфейс під кожен пристрій окремо, буде достатньо створити прогресивний веб-додаток, який буде слугувати як веб-сайтом, так і мобільним та настільним додатком. [4]

Прогресивний веб-додаток Blazor WebAssembly має такі переваги як:

- один інтерфейс на сайт, настільний та мобільний додатки
- можливість використання в автономному режимі (офлайн додаток)
- можливість використання переваг звичайних веб-додатків, таких як кешування та адаптивний дизайн [5]

Для встановлення веб-сайту як додатку на ПК, достатньо в адресній строці натиснути на «плюс», після чого додаток буде інстальовано як звичайну програму, яку можна буде знайти в панелі «пуск» або винести на робочий стіл.

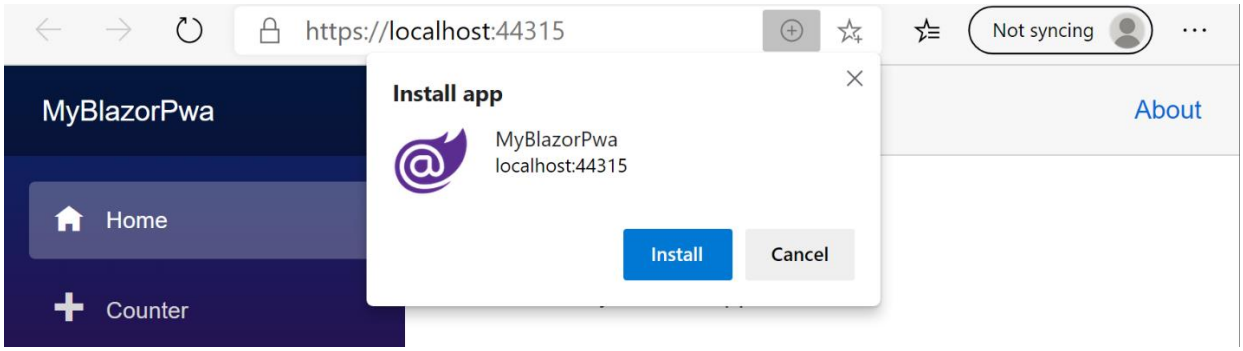



*Рисунок 4.1. Blazor додаток, інсталяція на ПК*

Подібним чином прогресивний веб-додаток Blazor WebAssembly можна встановити як звичаний додаток на мобільний телефон. Також присутня можливість завантаження такого додатку в Play Market та App Store.

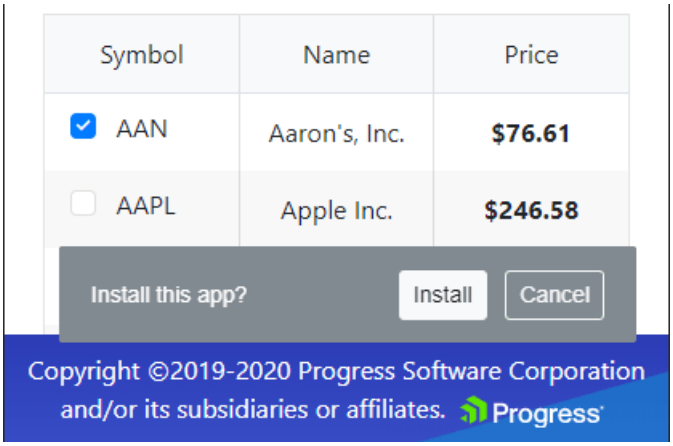



*Рисунок 4.2. Blazor додаток, інсталяція на мобільний пристрій*

Для взаємодії з сервером та базою даних використаємо описаний раніше фреймворк ASP.NET Web API.

## 4.3. Проектування бази даних

Зважаючи на опис запропонованої системи, в базі даних потрібні будуть таблиці для збереження інформації про авторизацію, житлові приміщення (квартири, будинки), діалоги між користувачами, договори оренди та місця, по яким буде виконуватись пошук житла, зокрема метро.

Для авторизації буде використовуватись BankID, проте інформація про авторизацію буде зберігатись саме в базі даних запропонованого веб-сервісу. Для зручного управління інформацією про авторизацію використаний фреймворк ASP.NET Identity. Застосовуючи його разом з нашою базою даних, він створює набір таблиць, якими буде управляти серверна частина додатку для ідентифікації того чи іншого користувача.

Житлові приміщення будуть розділені на декілька таблиць для більш швидкого пошуку по ним. Також буде створена таблиця, яка буде реалізовувати зв’язок один до одного з таблицями типів житла. Це допоможе зробити зручний пошук по інформації, яка узагальнена для всіх типів квартир та будинків. Зокрема такою інформацією є власник, адреса, широта та довгота, кількість кімнат, ціна та вказаний період здачі житла.

До таблиці з житловими приміщеннями реалізований зв’язок з таблицею договорів оренди житла. Дана таблиця буде містити відповідні ключі до конкретного житлового приміщення та користувача сервісу, що допоможе встановити зв’язок усіх орендованих та зданих квартир / будинків.

Таблиці, які відносяться до зберігання інформації діалогів зпроектовані таким чином, щоб не виникало конфліктних ситуацій саме в базі даних. Щоб не створювати таблицю діалогів з подвійним зв’язком до користувача сервісу для формування двох сторін діалогу, було створено додаткову таблицю, таким чином реалізувавши відношення багато діалогів до багатьох користувачів. Таблиця з самими повідомленнями же містить інформацію про текст повідомлення, час відправлення та самого відправника повідомлення в заданий діалог.

Інформацію про місця пошуку, зокрема інформацію про метро, було винесено у відповідні таблиці. Збереження інформації спроектоване в базі даних таким чином, щоб здійснювати доступ до записів таблиць було зручно. Кожна таблиця типізованої інформації – метро та парк, має зовнішній ключ, який вказує на таблицю з загальною інформацією. До того ж створена проміжна таблиця, яка поєднує усі місця (метро) з заданими в сервісі житловими приміщеннями таким чином, щоб відстань між ними була заданої відстані. Зроблено це для того, щоб не виконувати операцію з усіма квартирами та місцями для пошуку кожного разу при пошуковому запиті, а зберігати цю інформацію окремо і користуватись нею у випадку пошуку. Це значно зекономить час пошуку та навантаження на сервер, при цьому зайнявши лише місця для ключей, які поєднують інформацію про житлові приміщення та місця пошуку.

Окремою є таблиця для міграцій бази даних, вона містить інформацію про усі дії проведені відносно зміни бази даних.

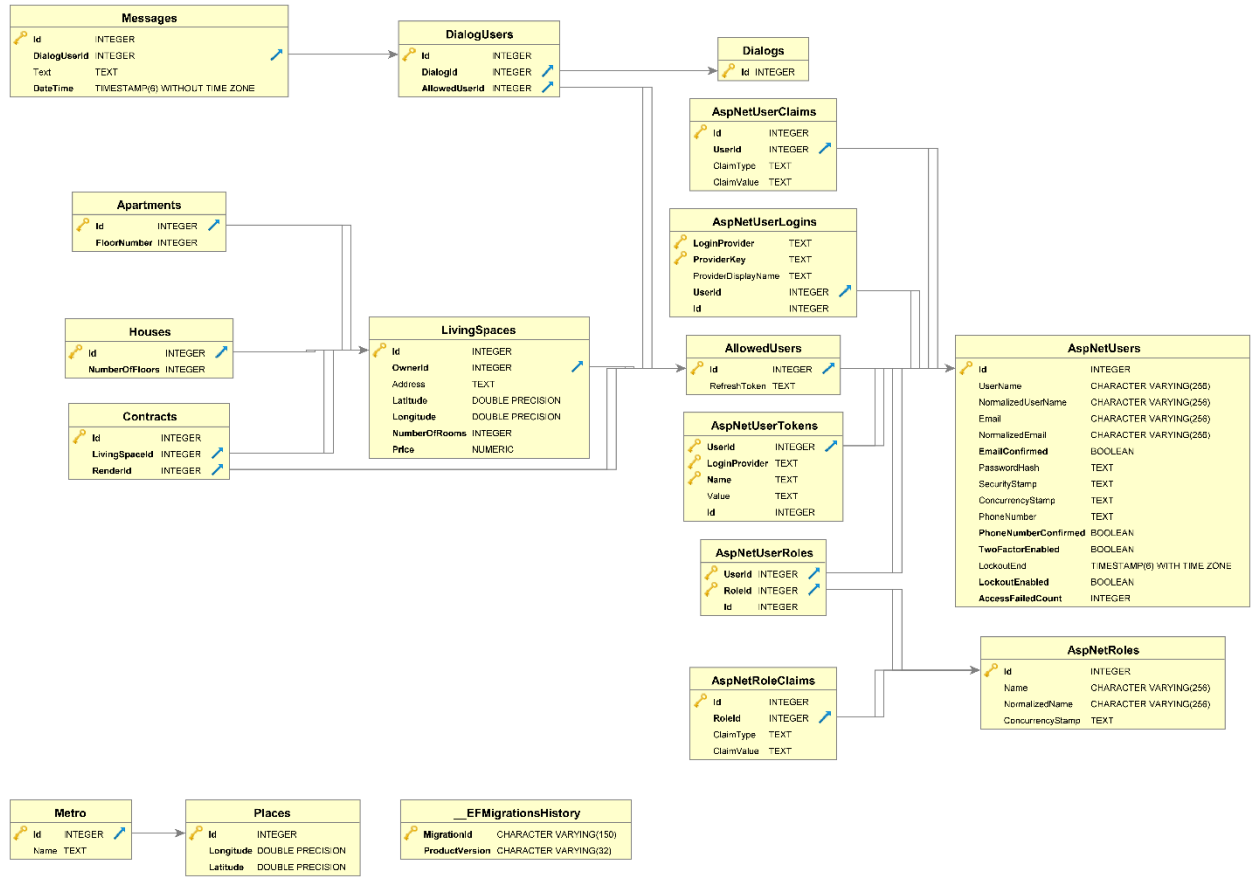



*Рисунок 4.3. Схема бази даних*

## 4.2. Розробка серверної частини веб-сервісу

Серверна частина веб-сервісу буде реалізована у вигляді API (прикладний програмний інтерфейс) до якого будуть надходити запити на отримання, зміну, видалення інформації. API надає відповідний інтерфейс, який користувачі зможуть використовувати.

### 4.2.1. Опис головних запитів до API

Опишемо основні запити обміну даними між інтерфейсом веб-сайту, мобільного або настільного додатку та сервером відображені в таблиці:

1. GET /livingspaces/all/{query} – запит на отримання даних про житлові приміщення з фільтрами сформованими в рядок «query». Приклад значення «query»:

   type=house;minprice=7000;maxprice=10000

   На стороні сервера дана строка запиту легко розпаршується і у

відповідь надсилається список усіх житлових приміщень, які задовільняють даний запит.

2. GET /livingspaces/show/{id} – запит на отримання інформації про конкретне житлове приміщення, id якого задано параметром.
3. GET /livingspaces/contact/{id} – запит на отримання контакту (посилання на відповідний діалог) власника житла. Параметром id вказаний ідентифікатор самого житла.
4. POST /livingspaces/publish – запит на публікацію оголошення про доступне для оренди житло. Приймає JSON моделі, яка містить обов’язкові та додаткові поля про житлове приміщення. Разом з тим, за допомогою ASP.NET Identity та авторизаційного токену, сервер ідентифікує користувача, та встановлює зв’язок з опублікованим оголошенням та самим користувачем.
5. PUT /livingspaces/update/{id} – запит на зміну інформації про опубліковане оголошення житла. Приймає аналогічну JSON модель до тієї, яка передавалась для публікації житла та оновлює інформацію про житлове приміщення.
6. PUT /livingspaces/deactivate/{id} – запит на сховання житлового приміщення з списку доступних. Використовується для того, щоб навіть коли житлове приміщення не є орендованим, але власник житла не має бажання або потреби на даний час його здавати, житло не було опублікованим.
7. DELETE /livingspaces/published/{id} – запит на видалення раніше опублікованого житлового приміщення.
8. POST /livingspaces/activate/{id} – запит на показ раніше доданого та прихованого житла.
9. POST /livingspaces/rent/{id} – надсилає запит на оренду житла та надсилає перше повідомлення в діалог з власником житла. Параметр

Id – ідентифікатор житла.

10. POST /livingspaces/rent/accept/{id} – запит на погодження договору оренди житла зі сторони власника житла. Після цього усі раніше подані запити на дане житлове приміщення стають неактивними, а в моніторинг орендованих приміщень, та тих що здаються додається дане приміщення, відповідно для орендатора та власника житла.
11. POST /livingspaces/rate/{id} – запит разом з яким передається JSON модель з оцінкою та відгуком про житлове приміщення. Доступний лише поточному або минулому орендатору.
12. POST /users/rate/{id} – запит разом з яким передається JSON модель з оцінкою, відгуком та типом користувача (орендатором, власником житла). Доступний лише поточному або минулому орендатору / власнику житла.

Усі запити, до яких адміністратор звертається через його панель управління реалізовують CRUD модель запитів (створення, читання, оновлення, видалення), і є практично аналогічними. Усі запити мають однаковий відповідний шлях та відрізняються лише типом методу. Запит на створення відповідного об’єкту використовує метод POST, запит на оновлення – PUT, на видалення – DELETE. На отримання інформації користувач повинен використати метод GET.

### 4.2.2. Документація API проекту

Документація усіх запитів до прикладного інтерфейсу веб-сервісу реалізована за допомогою фреймворку Swagger, який аналізує код проекту та візуалізує усі http запити, які можуть бути надіслані до API.

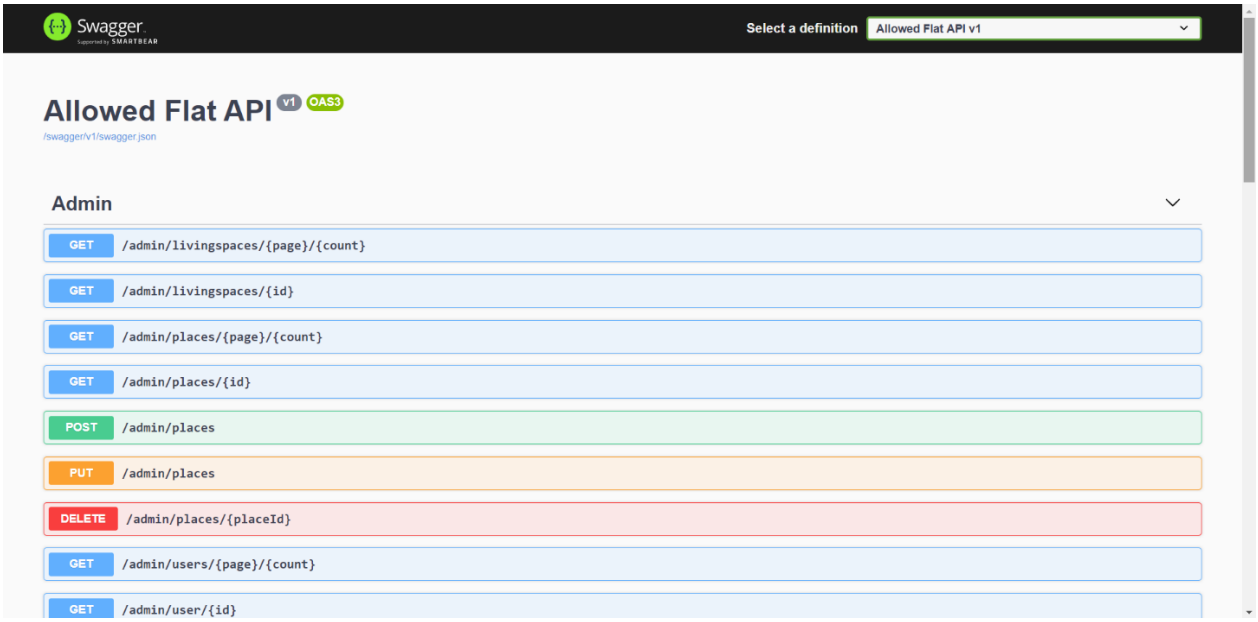



*Рисунок 4.4. Візуалізація запитів до API*

Додатково генерується json файл, де в відповідному форматі записані уся інформація про запити.

За допомогою даного фреймворку можна не лише переглянути шляхи до запитів, що вони приймають та що надсилають у відповідь, але й протестувати роботу API за допомогою веб-інтерфейсу фреймворка.

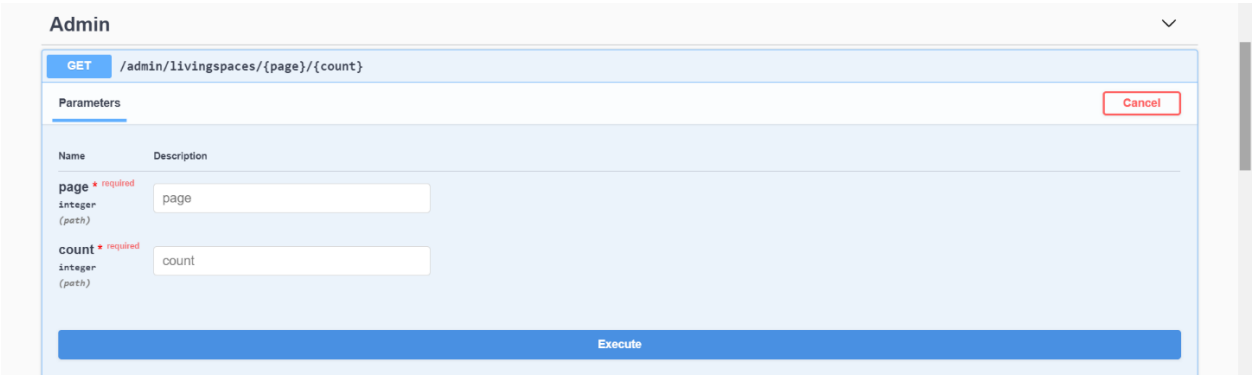



*Рисунок 4.5. Тестування роботи запиту до API*

При тестуванні розробник має можливіть передати відповідні параметри, обов'язкові з них позначені зірочкою та ключовим словом required та здійснити запит, отримавши на нього відповідь та статус код запиту.

Таким чином, реалізована мінімальна достатня документація для серверної частини проекту, що при розробці клієнтської частини допоможе

швидше реалізувати той чи інший запит та відобразити потрібну інформацію з серверу.

4.2.3. Опис взаємодії серверної частини сервісу з базою даних

Взаємодія серверної частини сервісу з базою даних відбувається за допомогою фреймворка Entity Framework Core, який має найбільшу популярність серед розробників додатків, хто використує мову програмування C# та .NET Core фреймворк. Зважаючи на те, що в якості системи управління базами даних була обрана об'єктно-реляційна система PostgreSQL, також було використано додатковий фреймворк Npgsql.EntityFrameworkCore.PostgreSQL, який розширює можливості попередньо названого фреймворка та надає можливість використовувати його у взаємодії з системами управління базами даних PostgreSQL.

Керуючись шаблоном проектування Dependency Injection, який в .NET Core фреймворку реалізований стандартно, було спроектовано роботу з базою даних через класи сервіси, які реалізовують відповідні інтерфейси. Даний підхід проектування надає можливість при потребі легко змінити систему управління базами даних або ж саму реалізацію запитів до бази даних не змінюючи інтерфейси роботи з ними. Також це є корисним при юніт тестуванні. Створюючи інтерфейси роботи з базою даних, ми надаємо можливість в подальшому застосовувати фреймворк Moq для тестування роботи сервісів без взаємодії з самою системою управління базами даних.

Опишемо приклад такого класу сервісу, який реалізовує додавання місць для пошуку (фільтрації) та обраховує всі дистанції між опублікованими житловими приміщеннями. Інтерфейс такого класу описує 4 базові функції управління даними (CRUD), такі як створення місця для пошуку, отримання інформації про нього, його редагування та видалення.

Додавання місця для пошуку враховує тип самого місця, тобто

спроектовано декілька методів інтерфейсу, таких як додавання метро або парки. Для кожного з типів місць є інформація, яка притаманна лише йому. Тобто при додаванні метро буде враховуватись колір гілки, при додаванні парку його не буде.

Редагування реалізоване подібним чином, при змінні типу місця, інформація, яка притаманна для усіх місць, які можна використовувати при пошуку, таких як назва, координати місця буде збережена, а типізована інформація, така як колір гілки метро буде замінена на інформацію відповідного типу.

Видалення відбувається за ідентифікаційним номером місця (id), так само як і отримання інформації про нього.

Також даний сервіс виконує обчислення відстаней між місцями для пошуку та вже опублікованими квартирами (аналогічна зворотня дія відбувається при публікації квартири користувачем). Дана функція реалізована таким чином, щоб навантаження на базу даних та кількість запитів до неї були мінімальними.

Для того, щоб не здійснювати запити до бази даних в циклі та проводити відносно громісткі обрахунки, що негативно вплине на її навантаження та швидкодію, було вирішено за допомогою методу AsEnumerable вивантажувати почергово кожне житлове приміщення в оперативну пам’ять та вже в ній проводити обчислення. Якщо дистанція між об’єктами менша заданої, тоді виконувати запис в базу нового об’єкту дистанції.

### 4.2.4. Модульне тестування коду

Модульне тестування (Unit тести) – це метод тестування, за допомогою якого, розробник може перевірити коректність роботи кожного написаного модуля коду. Більшість проектів потребують тестування, адже при зміні блоку коду, який може впливати на роботу інших частин програми, можуть

виникнути проблеми, віднайти які без системи тестувань буде проблематично, особливо це стосується об’ємних проектів, таких як запропонований сервіс пошуку оренди житла.

Для реалізації системи тестувань було обрано стандартний фреймворк, запропонований Visual Studio 2019 – Nunit. За домогою атрибута «Test», який вказує на метод для тестування та статичного класу Assert, який верифікує результат тесту було описано декілька тестів.

Опишемо один з методів, для якого був написаний тест. Даний метод обчислює відстань між об’єктами, нанесеними на карту, по координатам довготи та широти. Були реалізовані формули для обчислення в методі DistanceTo статичного класу CoordHelper. Після чого був написаний метод для тестування CalculateDistancesTest.

Для початку, метод тестування ініціалізує дані про місця, між якими буде знаходитись відстань. Після чого викликає метод, який обчислює відстань між ними. В результаті відбувається перевірка обчислень з обчисленнями, які були проведені за допомогою іншого сервісу або самостійно. В моєму випадку, я перевіряв декілька значень, обчислюючи їх самостійно та перевіряючи обчислення додатково за допомогою сервісу distance [10].

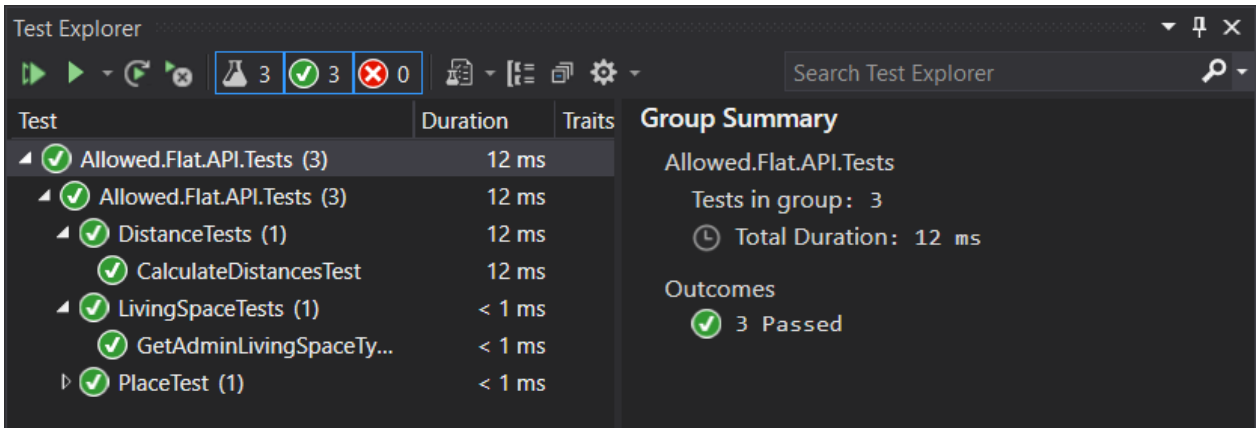



*Рисунок 4.6. Результати тестування*

4.2.4. Опис запитів в реальному часі

Усі запити на надсилання повідомлень та їх отримання відбувається у реальному часі за допомогою фреймворка SignalR, який використовує WebSocket де це можливо. При відсутності підтримки WebSocket, даний фреймворк використовує інші ефективні методи передачі інформації в реальному часі, такі як Long Polling. [6]

При встановлені з’єднання, сервер зберігає id користувача, та приймає запити на відправку повідомлення в відповідний діалог, та у випадку надсилання даному користувачу повідомлення надсилає запит з відповідними даними тексту повідомлення та дати надсилання. Усі дані передаються в форматі JSON, так як на жаль Blazor WebAssembly поки не підтримує більш компактний метод передачі даних – MessagePack.

Для того, щоб реалізувати функціонал чату між двома вибраними користувачами, на стороні сервера опишемо хаб (hub), який приймає запити з даними про повідомленнями (текст повідомлення та для кого воно) та одразу відправляє їх для користувача, який має отримати повідомлення. Для цього опишемо метод, який виконує даний функціонал, а також визначимо методи для підключення (OnConnectedAsync) та відключення (OnDisconnectedAsync) користувача від хабу, де будемо записувати ідентифікатор користувача (id) та видаляти його відповідно. Потрібно це для того, щоб ідентифікувати адресата і користувача, кому повідомлення було адресоване.

Для збереження ідентифікаторів користувача використовуватимемо словник (Dictionary), де ключем буде слугувати унікальний ідентифікатор підключення, який ми отримуємо з контексту хабу, а значенням буде нікнейм користувача.

Для зберігання повідомлень опишемо клас сервіс, який реалізовує патерн репозиторій та взаємодіє з базою данних. Записанні при надсиланні повідомлення будуть відповідати на запит їх отримання при першому звернені

до сторінки з діалогами. В даному випадку для передачі інформації ми використаємо звичайні http запити, так як вони будуть доволі громісткими та будуть суттєво впливати на швидкодію самого хабу.

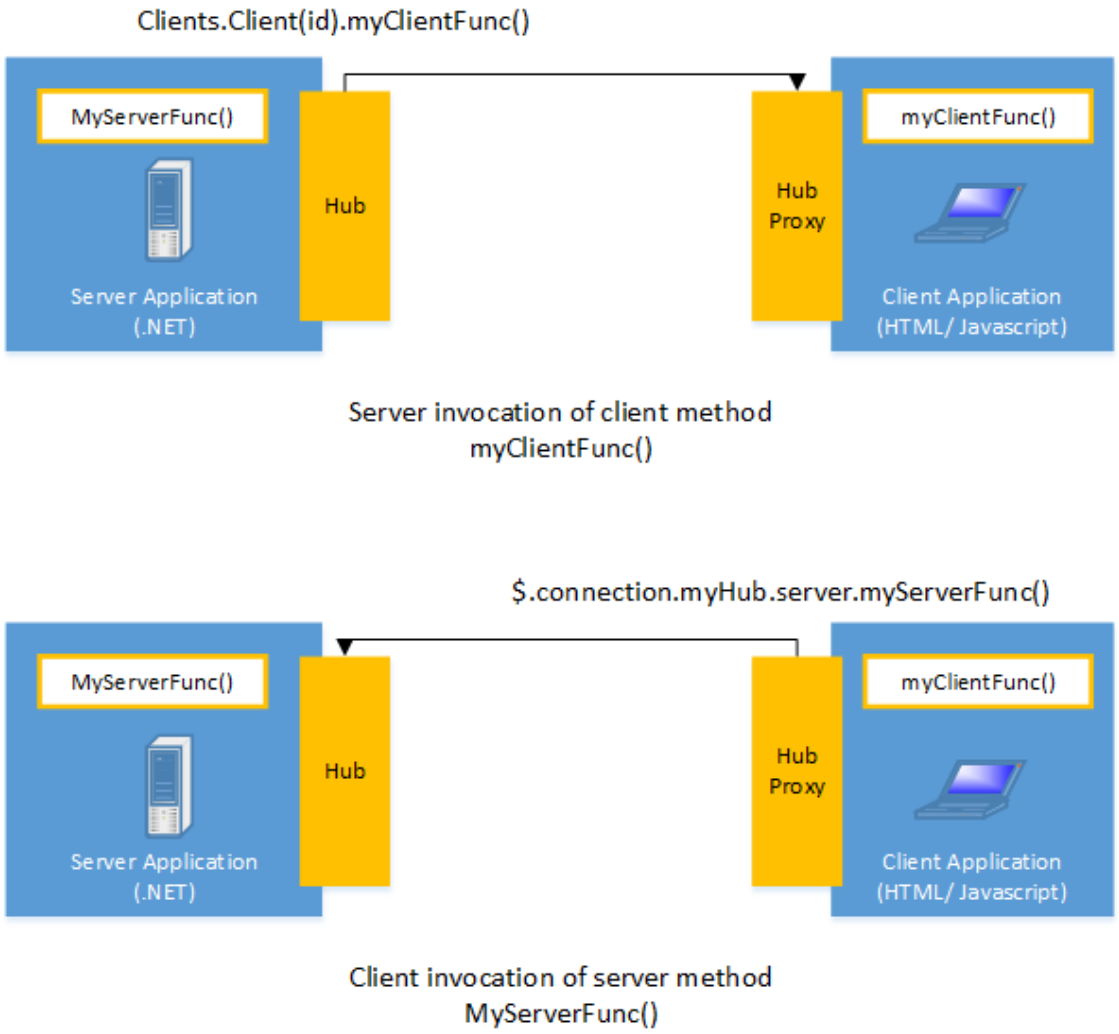



*Рисунок 4.7. Схема роботи SignalR*

Клієнтська частина сервісу буде реалізовувати сторінку з SignalR підключенням, за допомогою бібліотеки Microsoft.AspNetCore.SignalR.Client.

Створимо клас (ClientBase), який буде управляти підключеннями та у разі втрати з'єднання його відновлювати. Даний клас реалізовує метод підключення, створення нового підключення, перепідключення та відключення клієнта від хабу серверу. Реалізований функціонал можливо застосовувати не лише для написання повідомлень, але й в подальшому для

реалізацій запитів у реальному часі.

Для визначення методів отримання та надсилання повідомлення опишемо наступний клас – DialogClient, який наслідуватиметься від BaseClient, який в свою чергу реалізовує весь функціонал пов’язаний з підключеннями. Після чого нам залишається лише відобразити для користувача інформацію у вигляді html розмітки та пов’язати натиснення кнопки відправлення з методом відправки повідомлення.

### 4.3. Клієнтська частина додатку

#### 4.3.1. Реалізація запитів до API

На клієнтській частині веб-сервісу відбувається надсилання та отримання запитів з відображенням даних у html розмітці. Технологія WebAssembly дозволяє виконувати всі запити з C# коду, не використовуючи JavaScript.

Для зручності надсилання запитів та отримання відповідей на них був використаний фреймворк Refit. Він допомагає описувати надсилання та прийом запитів за допомогою атрибутів [Get], [Post], [Put], [Delete], замість того, щоб описувати окремі функції для цього за допомогою HttpClient, який використовується в Blazor для надсилання запитів замість аналога в JavaScript функції fetch.

Також Refit надає зручний функціонал обробки помилок, наприклад, 401 Unauthorized. Використовуючи операнди try catch і об’єкт класу ApiException, який надає нам фреймворк Refit, є змога обробити помилку неавторизованого користувача та сповістити його про необхідність авторизації. Дана функція також корисна для оновлення токену авторизації. Якщо термін дії токену вийшов, то надсилається запит на його оновлення. В разі успішного оновлення, користувач має змогу далі користуватись веб-сервісом в авторизованому вигляді.

4.3.2. Опис нанесення маркерів на карту

Карта з маркерами доступних для оренди житлових приміщень реалізована за допомогою бібліотеки leafletjs.

Leafletjs – це бібліотека JavaScript з відкритим кодом для інтерактивних карт, зручних для мобільних пристроїв. Має функціонал картографування, який більшість розробників коли-небудь потребують. Даний фреймворк дозволяє відобразити карту всього світу абсолютно безкоштовно, на відміну від дорогого аналогу – Google Maps. [7]

Центрування показу було реалізовано відносно поточного розташування користувача за допомогою стандартної функції об’єкту navigator. Інтерфейс Navigator представляє стан і особливості користувацького агенту. Це дозволяє JavaScript скриптам віднаходити положення користувача в даний момент часу. [8]

Маркери нанесені на карту були створені в редакторі Paint.NET та заповнені відповідними кольорами (зелений, жовтий та червоний). Саме нанесення відбулось через функцію «L.Market» з параметром icon, який відповідає за кастомізацію нанесеного маркеру. По замовчуванню його колір є синім. Також на кожен маркер встановлена подія обробки кліка користувача. При натисканні на маркер, користувач зможе не переадресовуючись на саму сторінку житлового приміщення передивитись його фотографії та ціну.

4.3.3. Оптимізація клієнтської частини додатку

Не мінімізовані css стилі та js скрипти погано впливають на швидкодію сайту. Значно краще, якщо вони позбавлені пробілів, а назви локальних змінних скорочені. Тоді розмір таких файлів є значно меншим та передача їх до користувача відбувається швидше.

Також немало важливо мінімізувати кількість файлів, які підключаються до сайту. Це вплине не лише на його швидкодію, але й забезбечить зручний контроль версій даних файлів. При внесенні змін в файл

зі стилями або скрипт, користувачеві потрібно очищувати кеш браузера, щоб їх відобразити. Тому хорошою практикою є дописування до назв файлів їх версію. Для того, щоб не міняти версію кожного зміненого файла, є можливість скласти дані файли в один (бандл), та міняти версію для нього. Виключенням є стороні бібліотеки, які ми підключаємо. Зміни для них вносяться доволі рідко, тому щоб користувачеві не завантажувати їх при зміні файлів сайту, корисно їх винести в окремий бандл.

Реалізовувати поставлені дії вручну не є раціональним рішенням. Тому задля забезпечення автоматизації даної задачі використаєм доповнення до Visual Studio під назвою «Bundler & Minifier» [12]. Створюючи конфігураційний файл, де описані стилі та скрипти сайту, які потрібно винести в окремий бандл, налаштуємо збірку даних файлів при компіляції проекту.

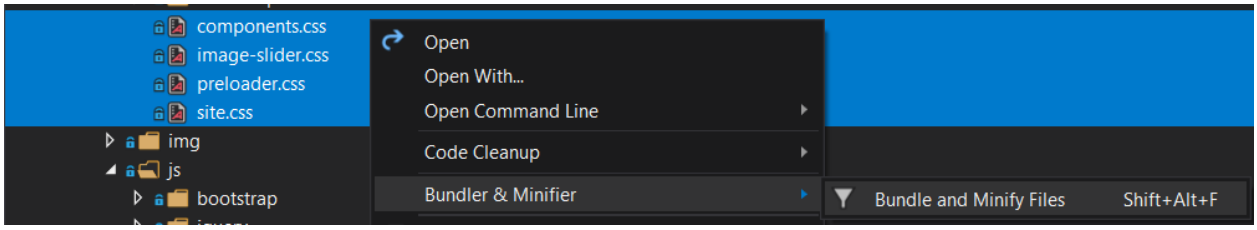



*Рисунок 4.8. Створення бандлу з стилів сайту*

Особливості фреймворку Blazor WebAssembly впливають на перше завантаження сайту. Тому також поставлена задачі оптимізації даної проблеми. Для її вирішення було знайдено бібліотеку під назвою Blazor Lazy Loading [11], яка підгружає бібліотеки фреймворку за потребою, замість того, щоб завантажувати всі одразу. Використовуючи дану реалізацію замінимо стандартний компонент загрузки Blazor – Router на відповідний до документації – LazyRouter. А також задамо компонент (preloader), який буде відображатись користувачеві до повноцінного завантаження сайту.

## 4.4. Технічна архітектура системи BankID

### 4.4.1. Опис принципу авторизації

Центральний вузол системи BankID НБУ взаємодіє з запропонованим

веб-сервісом на базі протоколу OAuth 2.0.

OAuth 2.0 дозволяє сторонньому додатку отримувати обмежений доступ до сервісу та взаємодіяти один між одним. Дана специфікація замінює собою застарілий протокол OAuth 1.0. [9]

Користувач ідентифікується за допомогою засобів АБС (автоматизована банківська система) відповідного банку чи його сервіс-провайдера, який представляє інтереси банку. Усі передані дані, які використовуються в запиті АБС до веб-сервісу, шифруються симетричним ключем або ключем шифрування даних за алгоритмом, який є визначеним у ДСТУ ГОСТ 28147-2009. SSL-протокол – є основою передач цих даних. [3]

Для реєстрації потрібно надати в системі BankID DNS-адресу веб-сервісу (client_host) з якої будуть відсилатись запити на авторизацію в BankID, та адресу зворотнього виклику (callback_url). Після узгодження усіх юридичних питань, BankID НБУ видає адміністратору веб-сервісу ідентифікатори параметрів з'єднання (client_id, client_secret).

Client_id – унікальний ідентифікатор веб-сервісу, складається з 32-шістнадцяткових значень, поділених на групи дефісами. Приклад client_id: 95e4ba81-06ad-4e97-b9d9- 0728fbed074f.

Client_secret – унікальний ідентифікатор секрету веб-сервісу, складається з 32-шістнадцяткових значень. Приклад client_secret: 5d42123a80942fda030c893c951fc08.

Callback_url – веб-адреса, на яку буде переадресований користувач з BankID НБУ у разі винекнення проблем / помилок при авторизації. Приклад callback_url: https://flat.allowed.space/bankid/callback.

Client_host – доменне ім'я, з якого будуть посилатись запити на авторизацію BankID. Приклад client_host: https://flat.allowed.space/.

### 4.4.2. Процедура авторизації BankID

OAuth 2.0 авторизація виконується у два етапи:

1. Отримання коду авторизації (authorization_code) – параметр, який отримується при запиті на веб-адресу callback_url.

2. Отримання коду доступу (access_token). Отримується при запиті на адресу BankID НУБ на підставі коду авторизації.

Перший етап (отримання коду авторизації) відбувається під час переадресації користувача кнопкою «Увійти за допомогою BankID» на сторінці авторизації у такому форматі: https://id.bank.gov.ua/v1/bank/oauth2/authorize?response_type=code&client_id=client_id&redirect_uri=callback_url&state= , де

callback_url – сторінка, на яку буде надіслана відповідь запиту з кодом авторизації та здійснена переадресація користувача;

state – ідентифікатор сесії, який генерується в BankID НБУ під час запиту зі сторони веб-сервісу.

Після успішної автентифікації BankID НБУ виконує запит за посиланням наданим до BankID – redirect_url. Разом з цим відбувається перенаправлення користувача на сторінку веб-сервісу.

Другий етап заключається в отриманні коду доступу (токену). Після отримання коду авторизації, веб-сервіс надсилає POST запит до BankID НБУ на адресу https://id.bank.gov.ua/v1/bank/oauth2/token з тілом запиту в JSON форматі, яке містить змінні:

- grant_type – при першому зверненні значення цього параметру є «access_token», при оновленні токену значення змінюється на «refresh_token».
- code – авторизаційний код, отриманий у попередньому кроці.
- callback_url – посилання на сторінку, на яку користувач буде перенаправлений при виникнені помилки авторизації.
- refresh_token – необов’язковий параметр, який дозволяє сесії існувати довше, за рахунок оновлення токену доступу.

У відповідь на даний запит BankID НБУ надсилає відповідь в JSON форматі, де вказаний авторизаційний токен, токен оновлення та термін дії основного токену.

Розглянемо помилки, які можуть винукнити при авторизації за допомогою системи BankID.

Під час першого етапу авторизації можливі дві ситуації:

1. Веб-сервіс не вдалось ідентифікувати (сервіс не зареєстрований в системі BankID або деякий параметр не співпадає). В такому випадку помилку буде відображено на сторінці авторизації BankID.
2. Ідентифікація користувача пройшла успішно, проте виникла інша помилка. Користувач буде переадресований на адресу, яка була передана в параметрі redirect_url з такими параметрами запиту:

- error – один із визначених специфікацією OAuth 2.0 кодів помилки (серед них можливі: access_denied – заборонений запит, invalid_request – недопустимі або відсутні обов’язкові значення деякого параметру, unauthorized_client – код авторизації отримати неможливо)
- error_description – опис помилки, деталі для розробників
- state – ідентифікатор сесії

Під час другого етапу рекомендовано звертати увагу на коди помилок HTTP. Також інформація про помилки може передаватись в тілі відповіді JSON формату, яке складається з таких полів:

- error – один із визначених специфікацією OAuth 2.0 кодів помилки
- error_description – опис помилки, деталі для розробників
- code – значення коду авторизації (authorization_code)

Розглянемо процедуру отримання даних користувача.

Для того, щоб отримати дані про користувача, потрібно виконати запит з такими параметрами як: перелік даних, які потрібно отримати, посилений сертифікат шифрування в форматі base64. Також в заголовки (headers) запиту

потрібно передати код доступу (access_token) таким чином:

Authorization: “Bearer access_token”

Опишемо основну інформацію, яка доступна для запиту.

Блок інформації fields:

- lastName (Прізвище)
- firstName (Ім’я)
- middleName (По батькові)
- phone (Контактний номер телефону)
- birthDay (Дата народження)
- sex (Стать)
- email (Електронна адреса)

Блок інформації scans:

- type (Копії документів, серед яких допустимі значення: passport – паспорт, zpassport – закордонний паспорт, inn – ідентифікаційний код, personalPhoto – фотографія) [3]

Саме ці дані веб-сервіс запрошує у системі BankID. Блок інформації слугує для можливості зв’язку з користувачем поза сервісом, для повідомлення деяких ситуацій, а в разі неправомірних дій, використовується блок інформації scans, інформація з якого передається до правоохоронніх органів.

### 4.4.3. Збереження даних користувача в сервісі

Інформація, яка передається від BankID до запропонованого веб-сервісу пошуку оренди житла, є конфіденційною та потребує відповідного захисту при її збереженні. Для цього фотографію скану паспорту не можна зберігати як файл з відкритим до посиланню доступом, тому було реалізовано передачу даного зображення в форматі base64. В такому ж форматі фотографії зберігаються в базі даних.

Уся інформація про користувача також шифрується за допомогою

алгоритму Advanced Encryption Standard. База даних потребує додаткового захисту, тому даний алгоритм шифрування був застосований для конфіденційної інформації. Адже при отриманні файлу бази даних сторонніми особами, її легко буде зчитати та отримати всю потрібну інформацію. Зашифрована інформація ж потребуватиме ключ для розшифрування.

AES (Advanced Encryption Standard) – симетричний алгоритм блочного шифрування. Даний алгоритм може використовувати ключ шифрування на 256 байт, що забезпечує надійне шифрування даних. В проекті AES шифрування реалізоване за допомогою бібліотеки System.Security.Cryptography.

### 4.5. Система контролю версіями

Для керування версіями проекту було вибрано GitLab – систему керування репозиторіями програмного коду для Git. Даний вибір обґрунтовується безкоштовністю його використання в приватних репозиторіях та зручністю використання в програмному середовищі Visual Studio.

GitLab має розширення для Visual Studio під назвою GitLab Extension for Visual Studio [13]. За допомогою нього кожен розробник має можливість виконувати стандартні операції Git, такі як commit (запис в репозиторій), fetch (перегляд змін, внесених в репозиторій іншими розробниками), pull (зтягнення змін, внесених в репозиторій іншими розробниками), push (внесення змін у віддалений репозиторій). Також дане розширення має не стандартну команду sync (синхронізація змін, внесених розробниками), яка є об’єднанням команд pull та push.

Дане розширення також має зручний функціонал вирішення конфліктів, створення мерджів, нових гілок, їх видалення, проте в рамках написання дипломної роботи я використовував лише попередньо названі команди.

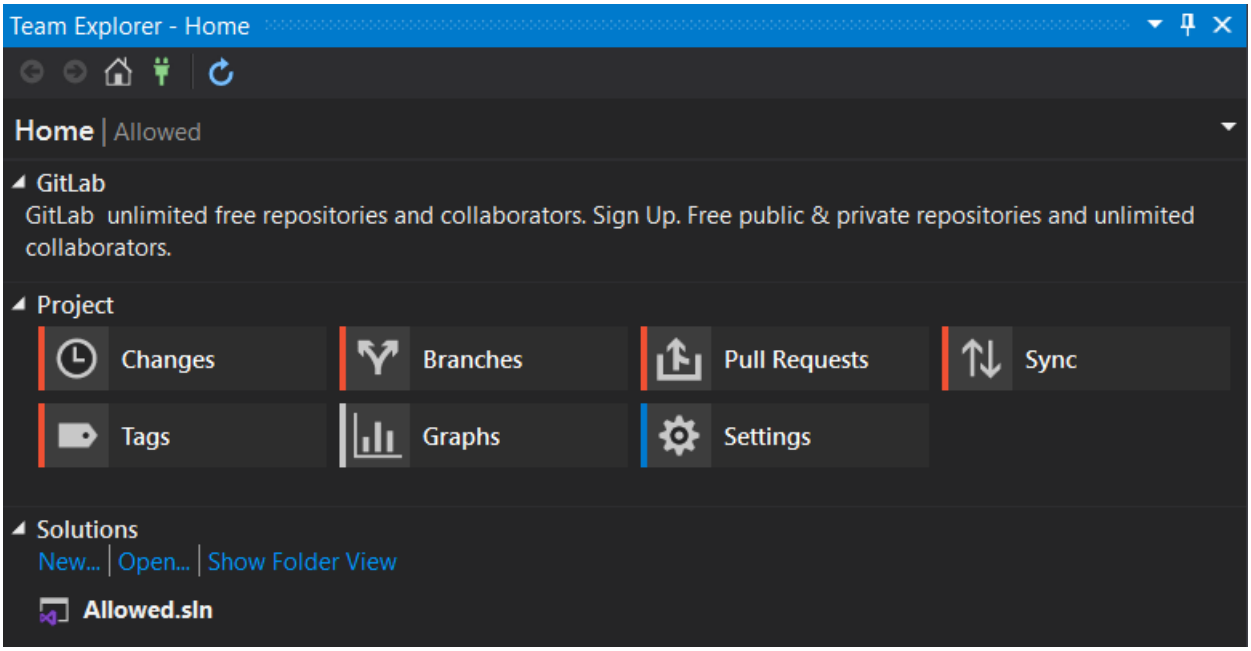



*Рисунок 4.9. Доповнення GitLab для Visual Studio 2019*

При публікації веб-сервісу на хостінг потрібно якимось чином відділяти код для представлення користувачам та код для розробки, для цього в проекті було створено дві додаткові гілки dev та prod відповідно.

Розділення на різні гілки особливо може бути корисним при використання TeamCity – система, яка дозволяє публікувати веб-сервіс при використанні команди push. При розділені гілок, оновлення не будуть публікувались одразу для користувачів, а публікація буде відбуватись на розробницьку версію веб-сервісу. В рамках даної дипломної роботи публікація на хостінг розглядатись не буде, проте в перспективі даний функціонал буде корисним, тому таке проектування розподілу гілок є раціональним.

## 4.6. Інструкція користувача

Після реалізації веб-сервісу потрібно забезпечити користувача інструкцією по його використанню. Програмний інтерфейс є доволі зрозумілим, проте опишемо найскладніші моменти користування веб-сервісом, робота з якими може викликати труднощі.

Одним з таких є авторизація за допомогою BankID. Розглянемо

авторизацію за допомогою державного банку ПриватБанк. Для авторизації за допомогою цього банку, потрібно бути зареєстрованим у системі Приват24.

Якщо користувач зареєстрований в данній системі, тоді при переадресації на сторінку вибору банку, він обирає банк, яким він користується (у нашому випадку Приват 24).

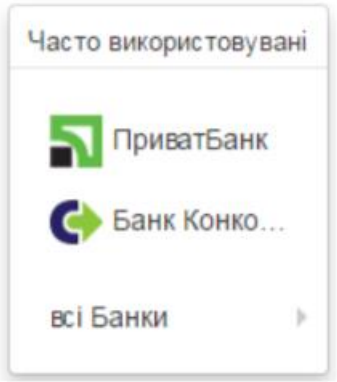



*Рисунок 4.10. Вибір банку для авторизації*

Далі потрібно авторизуватись у вибраному банку. У випадку Приват24 потрібно ввести номер телефону та пароль, після чого підтвердити всіх за допомогою СМС, яке буде надіслане на вказаний номер телефону.


*Рисунок 4.11. Процес авторизації в системі Приват24*

Якщо користувач не зареєстрований, для такого потрібно нажати на

кнопку «Зареєструватись», яка вказана нижче.

Обираючи будь-який інший банк процес авторизації буде відбуватись аналогічно, проте можуть бути і альтернативні способи авторизації. Як ми можемо побачити на рисунку 4.11, ПриватБанк надає нам можливість авторизуватись за допомогою QR-коду, який в свою чергу можна зчитати з додатку банку. Так чи інакше, усі українські банки обладнані надійною системою авторизації. Часто вона є двофакторною, на прикладі ПриватБанку, як описувалось вище, другим фактором авторизації може бути СМС повідомлення.

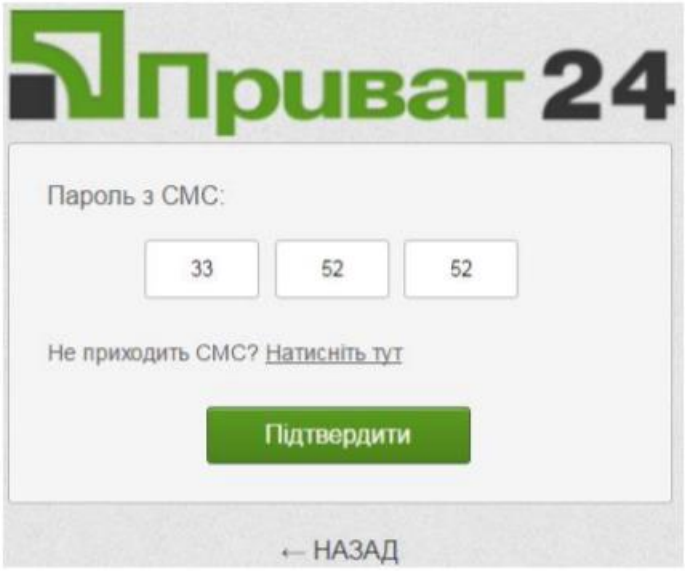



*Рисунок 4.12. Введення коду підтвердження, надісланого СМС повідомленням*

Окрім авторизації, труднощі можуть виникнути також з самим пошуком житла для оренди. Тому інтерфейс пошуку був розроблений максимально зручно для користувача.

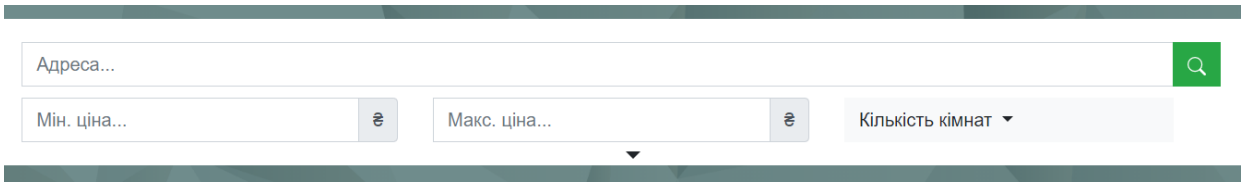



*Рисунок 4.13. Строка пошуку житла та фільтри*

Як ми можемо побачити на рисунку 4.13., спочатку для користувача

доступна лише строка для вводу адреси та місць для пошуку (метро, парки). Також виділені основні фільтри. Часто саме ці критерії пошуку є пріоритетними при виборі квартири або будинку для оренди. Вказавши дані по фільтрам та адресі користувач може натиснути на кнопку пошуку, після чого на карті з'являться маркери зеленого, оранжевого та червоного кольору, як це показано на рисунку 3.4. Колір даних маркерів залежить від відповідності (релевантності) значень фільтрів пошуку до існуючих житлових приміщень. Зручність таких позначень вже обґрунтовувалась у попередньому розділі, тому розглянемо лише процес користування даними маркерами.

Веб-сервіс пропонує користувачу спершу переглянути житлові приміщення, які мають зелений колір, як найбільш релевантні, далі оранжевий, далі червоний, проте користувач може натиснути на будь який з них, після чого з'явиться вікно з фотографіями даного приміщення, які можна гортати за допомогою слайдера, якщо таких фотографій більше однієї. Після їх перегляну користувач має можливість перейти на сторінку з більш детальною інформацією.

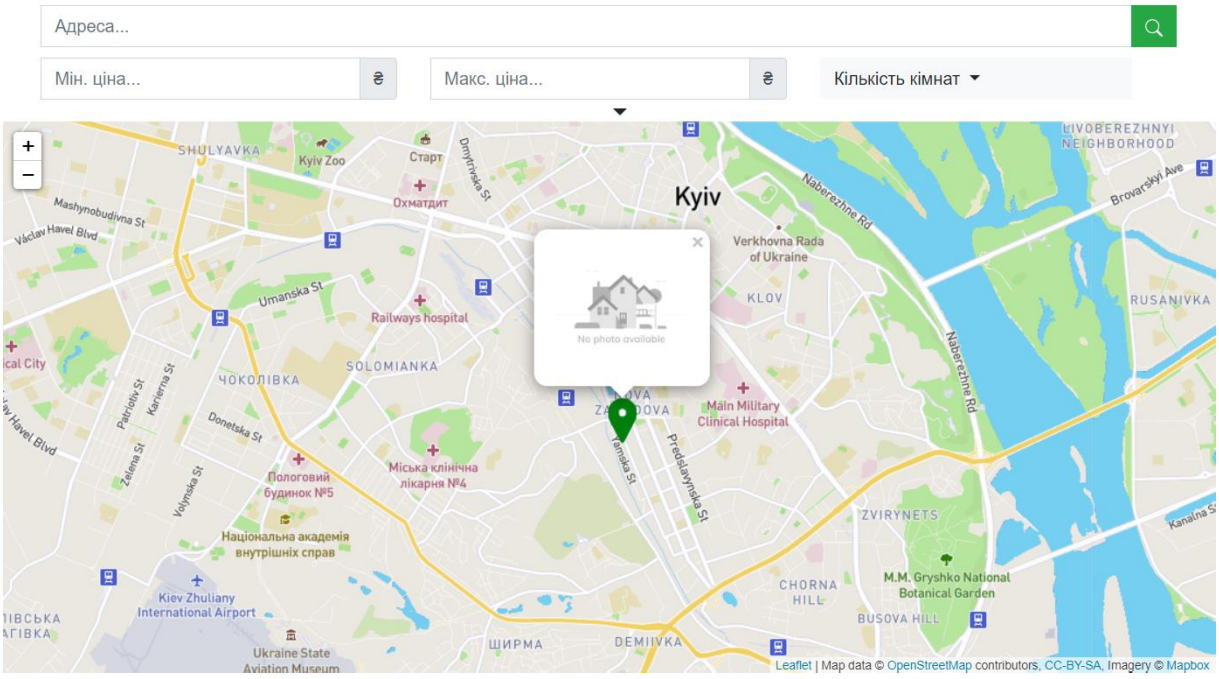



*Рисунок 4.14. Відображення фотографій над маркером на карті*

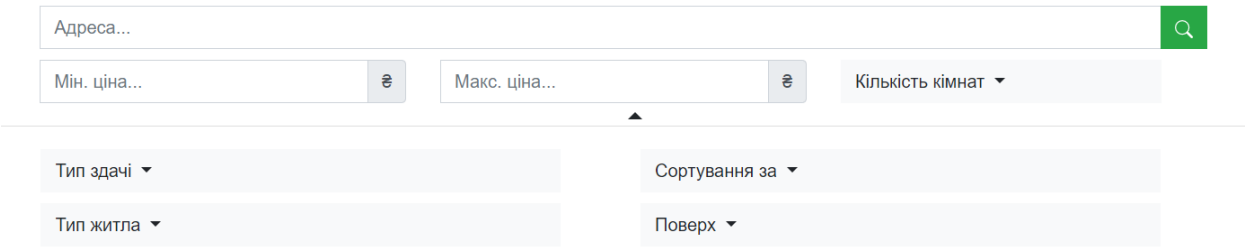



*Рисунок 4.15. Додаткові фільтри*

Користувач також має можливість виставити додаткові фільтри пошуку. Такими фільтрами є тип здачі (подобово, довгостроково), сортування за ціною або рейтингом, а також тип житла, він якого залежить останній фільтр. На момент написання дипломної роботи, користувач має змогу вибрати тип житла квартиру або будинок, залежно від того останній фільтр матиме значення «поверх» або «поверховість».

# ВИСНОВКИ ДО РОЗДІЛУ

В даному розділі було розглянуто та описано програмне рішення запропонованого веб-сервісу. Описані усі фреймворки та додаткові технології, які використовуються під час розробки системи. Також був описаний веб-інтерфейс сервісу, та його адаптація за допомогою фреймворку Blazor WebAssembly на мобільні та настільні додатки.

Спроектовано базу даних, описані всі таблиці та обґрунтовані усі зв’язки між ними. Були розглянуті основні функції серверної частини веб-сервісу, можливості користувачів як в ролі орендатора, так і в ролі користувача, описані обмеження для двох ролей.

Проаналізовано принцип роботи технології OAuth 2.0, а також роботу системи авторизації BankID. Розглянуті основні запити даної системи на отримання кодів доступу (токенів) авторизації та параметри, які передаються та приймаються під час їх виконання. Складена інструкція користувача для авторизації за допомогою системи BankID.

# ЗАГАЛЬНІ ВИСНОВКИ

Під час виконання дипломного проекту був розроблений веб-сервіс, який вирішує проблему безпеки та ідентифікацію особистостей під час пошуку та оренди житлових приміщень. Були переглянуті та проаналізовані можливі веб-сервіси аналоги, враховані їх недоліки та переваги. Даний сервіс має ряд іноваційних рішень по відношенню до них, таких як сортування з пріоритетним місцерозташуванням, а не критерією сортування, аналіз релевантності результатів пошукового запиту з врахуванням оцінок користувачів сервісу.

Були описані всі алгоритми та реалізації необхідного для веб-сервісу пошуку житла функціоналу, такі як: пошук по заданим фільтрам, відображення результатів пошуку на карті в сортованому вигляді, публікація нових оголошень, моніторинг орендованих та зданих користувачами сервісу квартир та будинків, меседжер для комунікації в межах сервісу.

Була реалізована авторизація на основі системи BankID, розглянуті основні принципи роботи даної системи, інформацію, яку вона може надавати, способи автентифікації через різні українські банки. Розглянуто технологію авторизації OAuth 2.0, яка використовується в BankID.

Для реалізації проекту була вибрана мова програмування, яка найбільш відповідала вимогам реалізації – C#. Дана мова була використана при розробці як клієнтської частини веб-сервісу так і серверної. Відповідно до неї був використаний перелік фремворків.

Проект розроблявся з використанням системи контролю коду GitLab, був проведений розподіл на гілку для розробки та тестування та на гілку публікації веб-сервісу з перспективою на використання системи оновлень TeamCity. Також проект був оптимізований за допомогою різних практик, включаючи мінімізацію файлів та враховуючи особливості застосованих

фреймворків. Для зручності розробки веб-сервісу на різні платформи всю серверну частину коду було задокументовано за допомогою фреймворку Swagger.

Реалізований веб-сервіс є безпечною та зручною реалізацією поставленої задачі розробки сервісу пошуку та оренди житла, яка може конкурувати з її українськими та іноземними аналогами.

# ПЕРЕЛІК ПОСИЛАНЬ


1. BankID [Електронний ресурс] – Режим доступу до ресурсу: https://bankid.org.ua/
2. Національний банк України [Електронний ресурс] – Режим доступу до ресурсу: https://id.bank.gov.ua/
3. Специфікація взаємодії з системою BankID Національного банку України (взаємодія системи BankID НБУ з порталом послуг) [Електронний ресурс] – Режим доступу до ресурсу: https://id.bank.gov.ua/assets/docs/Specification_BankID_NBU_Portal_v4.pdf
4. Build Progressive Web Applications with ASP.NET Core Blazor WebAssembly [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/core/blazor/progressive-web-app?view=aspnetcore-3.1&tabs=visual-studio
5. Telerik UI for Blazor in a PWA – Easy as Pie [Електронний ресурс] – Режим доступу до ресурсу: https://www.telerik.com/blogs/telerik-ui-for-blazor-in-a-pwa
6. Introduction to SignalR [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/signalr/overview/getting-started/introduction-to-signalr
7. Leafletjs [Електронний ресурс] – Режим доступу до ресурсу: https://leafletjs.com/
8. Navigator [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Navigator
9. The OAuth 2.0 Authorization Framework [Електронний ресурс] – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6749
10. Distance To [Електронний ресурс] – Режим доступу до ресурсу: https://ru.distance.to/
11. Blazor Lazy Loading [Електронний ресурс] – Режим доступу до ресурсу:

https://github.com/isc30/blazor-lazy-loading

12. Bundler & Minifier [Електронний ресурс] – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=MadsKristensen.BundlerMinifier
13. GitLab Extension for Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=MysticBoy.GitLabExtensionforVisualStudio
14. Інструкція щодо авторизації користувачів для використання електронних сервісів [Електронний ресурс] – Режим доступу до ресурсу: https://www.city-adm.lviv.ua/lmr/lmrdownloads/forms/8_-Avtoryzatsiia-korystuvacha_1.pdf

ДОДАТОК 1

Веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості

Схема бази даних

*ІАЛЦ.467800.004 Д1*

Аркушів 1

Київ 2020

**public.__EFMigrationsHistory**

| | | |
|---|---|---|
| MigrationId | character | PK |
| ProductVersion | character | |

**public.AspNetUserClaims**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| UserId | integer | | FK |
| ClaimType | text | NULL | |
| ClaimValue | text | NULL | |

**public.AspNetUsers**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| UserName | character | NULL | |
| NormalizedUserName | character | NULL | AK |
| Email | character | NULL | |
| NormalizedEmail | character | NULL | |
| EmailConfirmed | boolean | | |
| PasswordHash | text | NULL | |
| SecurityStamp | text | NULL | |
| ConcurrencyStamp | text | NULL | |
| PhoneNumber | text | NULL | |
| PhoneNumberConfirmed | boolean | | |
| TwoFactorEnabled | boolean | | |
| LockoutEnd | timestamp | NULL | |
| LockoutEnabled | boolean | | |
| AccessFailedCount | integer | | |

**public.AspNetUserLogins**

| | | | |
|---|---|---|---|
| LoginProvider | text | | PK |
| ProviderKey | text | | PK |
| ProviderDisplayName | text | NULL | |
| UserId | integer | | FK |
| Id | integer | | |

**public.AllowedUsers**

| | | | |
|---|---|---|---|
| Id | integer | | PK FK |
| FirstName | text | NULL | |
| LastName | text | NULL | |
| MiddleName | text | NULL | |
| DateOfBirth | timestamp | NULL | |
| Sex | boolean | NULL | |
| Passport | text | NULL | |
| RefreshToken | text | NULL | |

**public.AspNetUserRoles**

| | | |
|---|---|---|
| UserId | integer | PK FK |
| RoleId | integer | PK FK |
| Id | integer | |

**public.AspNetRoles**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| Name | character | NULL | |
| NormalizedName | character | NULL | AK |
| ConcurrencyStamp | text | NULL | |

**public.AspNetUserTokens**

| | | | |
|---|---|---|---|
| UserId | integer | | PK FK |
| LoginProvider | text | | PK |
| Name | text | | PK |
| Value | text | NULL | |
| Id | integer | | |

**public.AspNetRoleClaims**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| RoleId | integer | | FK |
| ClaimType | text | NULL | |
| ClaimValue | text | NULL | |

**public.Contracts**

| | | |
|---|---|---|
| Id | integer | PK |
| LivingSpaceId | integer | FK |
| RenderId | integer | FK |
| IsActive | boolean | |

**public.Dialogs**

| | | |
|---|---|---|
| Id | integer | PK |

**public.DialogUsers**

| | | |
|---|---|---|
| Id | integer | PK |
| DialogId | integer | FK |
| AllowedUserId | integer | FK |

**public.Messages**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| DialogUserId | integer | | FK |
| Text | text | NULL | |
| DateTime | timestamp | | |

**public.LivingSpaces**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| OwnerId | integer | | FK |
| Longitude | double | | |
| Latitude | double | | |
| Price | numeric | | |
| Address | text | NULL | |
| NumberOfRooms | integer | | |
| Type | text | NULL | |
| Description | text | NULL | |
| Status | text | NULL | |

**public.PlaceDistances**

| | | |
|---|---|---|
| Id | integer | PK |
| PlaceId | integer | FK |
| LivingSpaceId | integer | FK |
| Value | double | |

**public.Places**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| Longitude | double | | |
| Latitude | double | | |
| Name | text | NULL | |
| Type | text | NULL | |

**public.Metro**

| | | | |
|---|---|---|---|
| Id | integer | | PK FK |
| BranchColor | text | NULL | |

**public.LivingSpaceImages**

| | | | |
|---|---|---|---|
| Id | integer | | PK |
| LivingSpaceId | integer | | FK |
| Value | text | NULL | |

**public.Houses**

| | | |
|---|---|---|
| Id | integer | PK FK |
| NumberOfFloors | integer | |

**public.Apartments**

| | | |
|---|---|---|
| Id | integer | PK FK |
| FloorNumber | integer | |

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467800.004 Д1 | |
| Зм. | Арк. | № докум. | Підпис | Дата | Схема бази даних | Літ. / Маса / Масштаб |
| Розроб. | | Федорович В.Р | | | | |
| Перевір. | | Роковий О.П. | | | | Аркуш 1 / Аркушів 1 |
| Н. Контр. | | Сімоненко В.П. | | | Дипломна робота | НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІП-62 |
| Затв. | | | | | | |

ДОДАТОК 2

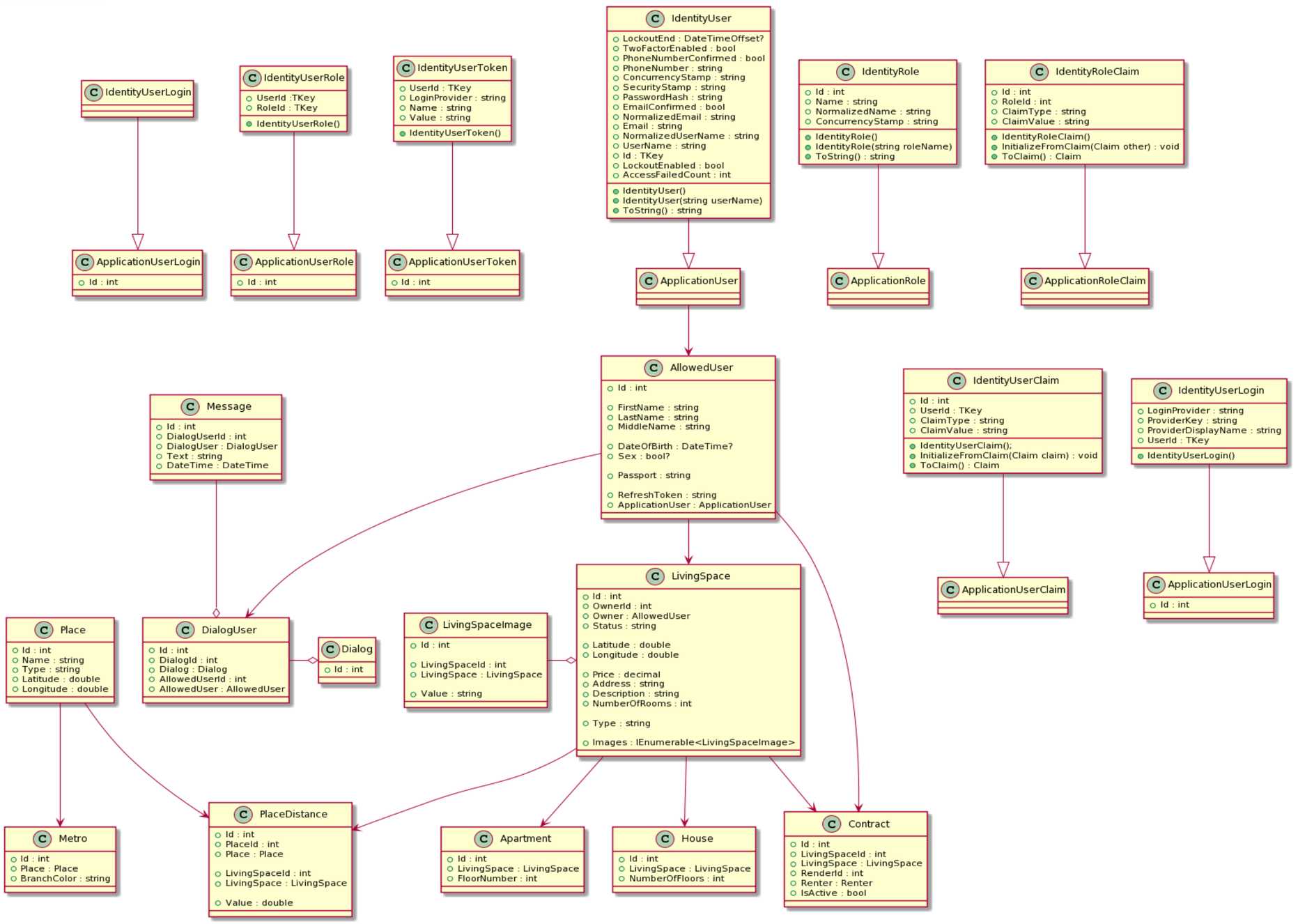
Веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості

Діаграма класів контексту бази даних

*ІАЛЦ.467800.005 Д2*

Аркушів 1

Київ 2020




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467800.005 Д2 | |
| Зм. | Арк. | № докум. | Підпис | Дата | Діаграма класів контексту бази даних | Літ. / Маса / Масштаб |
| Розроб. | | Федорович В.Р | | | | |
| Перевір. | | Роковий О.П. | | | | Аркуш 1 / Аркушів 1 |
| Н. Контр. | | Сімоненко В.П. | | | Дипломна робота | НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІП-62 |
| Затв. | | | | | | |

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № оригі.

# ДОДАТОК З

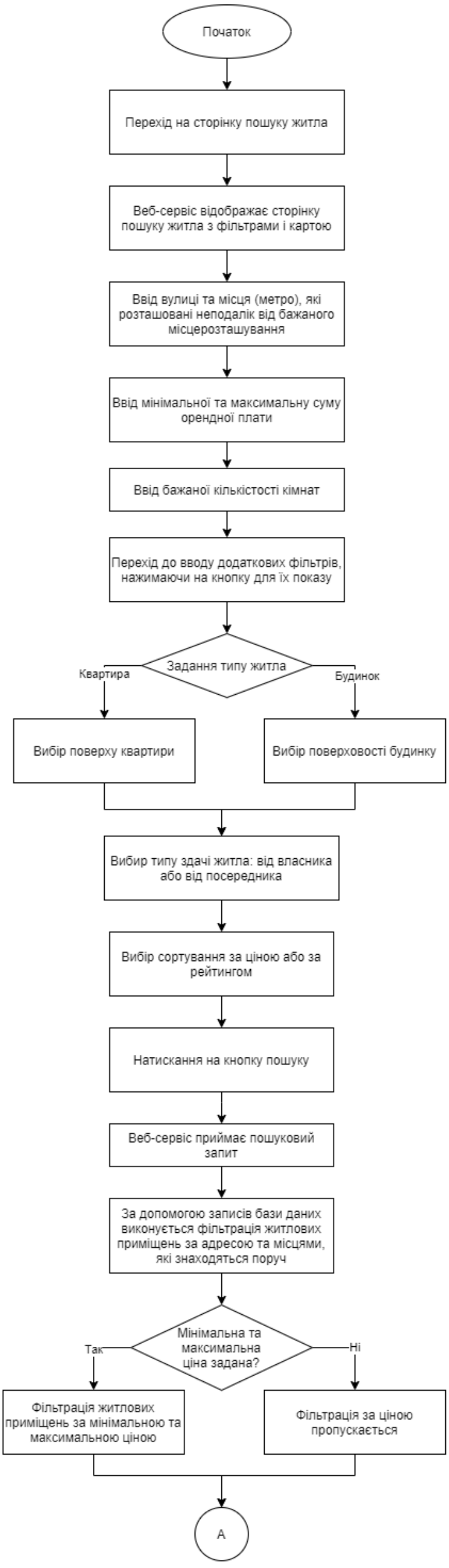
Веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості

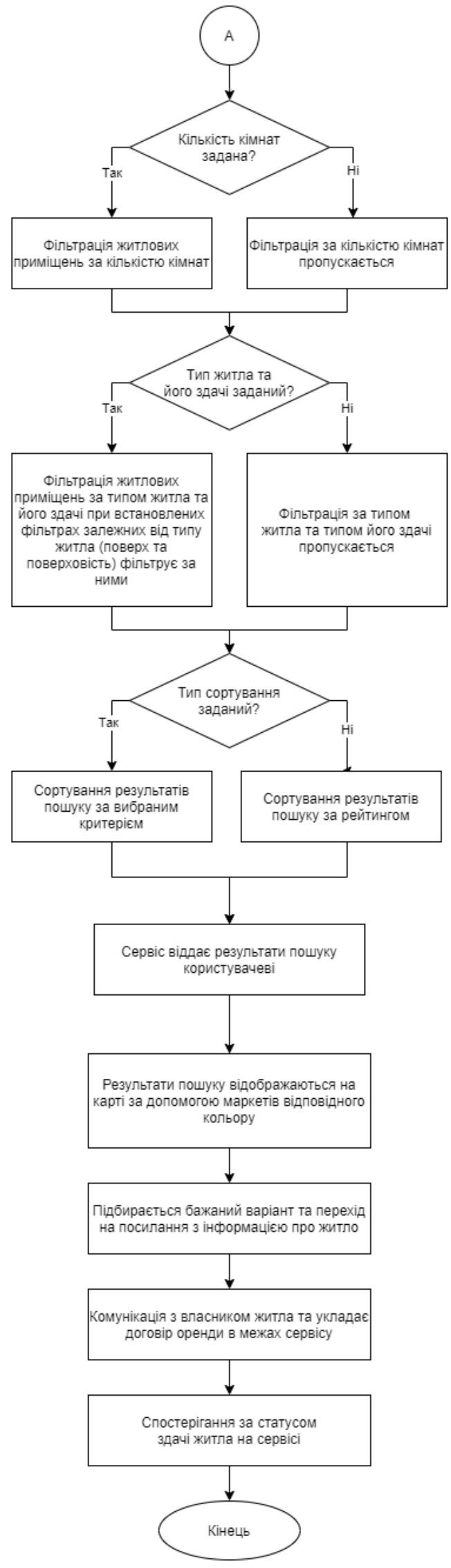
## Принципова схема пошуку житлових приміщень

*ІАЛЦ.467800.006 ДЗ*

Аркушів 1

Київ 2020




Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

| Зм. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.467800.006 ДЗ | | |
|---|---|---|---|---|---|---|---|
| Розроб. | | Федорович В.Р. | | | Принципова схема пошуку житлових приміщень | Літ. | Маса / Масштаб |
| Перевір. | | Роковий О.П. | | | | Аркуш 1 | Аркушів 1 |
| Н. Контр. | | Сімоненко В.П. | | | Дипломна робота | НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІП-62 | |
| Затв. | | | | | | | |

ДОДАТОК 4

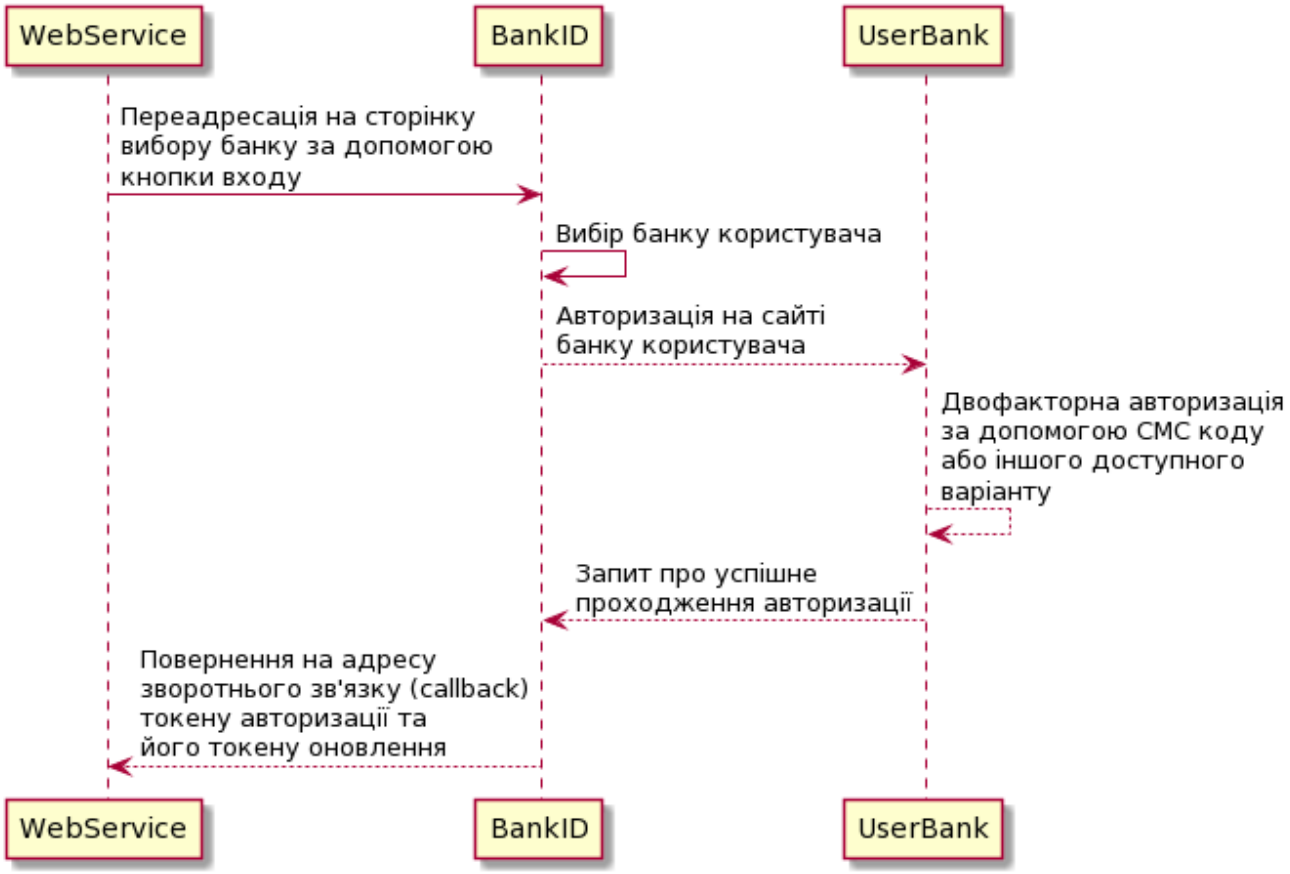
Веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості

Функціональна схема авторизації

*ІАЛЦ.467800.007 Д4*

Аркушів 1

Київ 2020




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | *ІАЛЦ.467800.007 Д4* | | |
| *Змн.* | *Арк.* | *№ докум.* | *Підпис* | *Дата* | | | |
| *Розроб.* | | *Федорович В.Р.* | | | *Функціональна схема авторизації* | *Літ.* | *Арк.* | *Аркушів* |
| *Перевір.* | | *Роковий О.П.* | | | | | *1* | *1* |
| | | | | | | *НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІП-62* | | |
| *Н. Контр.* | | *Сімоненко В.П.* | | | | | | |
| *Затверд.* | | | | | | | | |

ДОДАТОК 5

Веб-сервіс пошуку оренди житла з покращеною системою ідентифікації особистості

Текст програми

*ІАЛЦ.467800.008 Д5*

Аркушів 27

Київ 2020

AdminController.cs

```
using Allowed.Flat.API.Services.UserService;
using Allowed.Flat.Data.Constants;
using Allowed.Flat.Data.DbModels.LivingSpaces;
using Allowed.Flat.Data.DbModels.Places;
using Allowed.Flat.Data.DbModels.Users;
using Allowed.Flat.Data.Models.Admin.LivingSpaces;
using Allowed.Flat.Data.Models.Admin.Places;
using Allowed.Flat.Data.Models.Admin.Users;
using Allowed.Flat.Data.Models.Pagination;
using Allowed.Flat.Data.Models.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Linq;

namespace Allowed.Flat.API.Controllers
{
    [ApiController]
    [Route("admin")]
    [Authorize(Roles = "admin")]
    public class AdminController
    {
        private readonly IUserService _userService;
        private readonly IPlaceService _placeService;
        private readonly ILivingSpaceService _livingSpaceService;

        public AdminController(
            IUserService userService,
            IPlaceService placeService,
            ILivingSpaceService livingSpaceService)
        {
            _userService = userService;
            _placeService = placeService;
            _livingSpaceService = livingSpaceService;
        }

        #region LivingSpaces

        [HttpGet]
        [Route("livingspaces/{page}/{count}")]
        public PaginationList<AdminLivingSpaceDto> GetLivingSpaces(int page, int count)
        {
            return new PaginationList<AdminLivingSpaceDto>
            {
                List = _livingSpaceService.GetLivingSpaces((page - 1) * count, count)
                    .Select(l => new AdminLivingSpaceDto
                    {
                        Id = l.Id,
                        Address = l.Address,
                        Type = l.Type,
                        OwnerId = l.OwnerId,
                        OwnerName = $"{l.Owner.FirstName} {l.Owner.MiddleName}
{l.Owner.LastName}"
                    }).ToList(),
                CountAll = _userService.CountUsers()
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467800.008 Д5 | | |
| Змн. | Арк. | № докум. | Підпис | Дата | | | |
| Розроб. | | Федорович В.Р. | | | Функціональна схема авторизації | Літ. | Арк. 1 / Аркушів 27 |
| Перевір. | | Роковий О.П. | | | | | |
| Н. Контр. | | Сімоненко В.П. | | | | НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ ІП-62 | |
| Затверд. | | | | | | | |

```
            };
        }

        [HttpGet]
        [Route("livingspaces/{id}")]
        public AdminLivingSpaceDetailDto GetLivingSpace(int id)
        {
            LivingSpace livingSpace = _livingSpaceService.GetLivingSpace(id);

            return new AdminLivingSpaceDetailDto
            {
                Id = livingSpace.Id,
                Address = livingSpace.Address,
                Latitude = livingSpace.Latitude,
                Longitude = livingSpace.Longitude,
                Price = livingSpace.Price,
                OwnerId = livingSpace.OwnerId,
                OwnerName = $"{livingSpace.Owner.FirstName} {livingSpace.Owner.MiddleName}
{livingSpace.Owner.LastName}",
                NumberOfRooms = livingSpace.NumberOfRooms,
                Type = livingSpace.Type,
                TypedInfo = _livingSpaceService.GetTypedLivingSpace(id, livingSpace.Type)
            };
        }

        #endregion

        #region Places

        [HttpGet]
        [Route("places/{page}/{count}")]
        public PaginationList<AdminPlaceDto> GetPlaces(int page, int count)
        {
            return new PaginationList<AdminPlaceDto>
            {
                List = _placeService.GetPlaces((page - 1) * count, count)
                    .Select(p => new AdminPlaceDto
                    {
                        Id = p.Id,
                        Name = p.Name,
                        Type = p.Type
                    }).ToList(),
                CountAll = _userService.CountUsers()
            };
        }

        [HttpGet]
        [Route("places/{id}")]
        public AdminPlaceDetailDto GetPlace(int id)
        {
            Place place = _placeService.GetPlace(id);

            return new AdminPlaceDetailDto
            {
                Id = place.Id,
                Name = place.Name,
                Latitude = place.Latitude,
                Longitude = place.Longitude,
                Type = place.Type,
                TypedInfo = _placeService.GetTypedPlace(id, place.Type)
            };
```

```
        }

        [HttpPost]
        [Route("places")]
        public StatusModel AddPlace(AdminPlaceDetailDto place)
        {
            Place addedPlace = null;

            if (place.Type == AllowedConstants.Context.PlaceTypes.Metro)
            {
                AdminMetroDto metro =
JsonConvert.DeserializeObject<AdminMetroDto>(place.TypedInfo.ToString());

                addedPlace = _placeService.AddMetro(new Metro
                {
                    Place = new Place
                    {
                        Name = place.Name,
                        Latitude = place.Latitude,
                        Longitude = place.Longitude,
                        Type = place.Type
                    },
                    BranchColor = metro.BranchColor
                });
            }

            if(addedPlace != null)
            {
                _placeService.AddDistances(place.Id);
            }

            return new StatusModel { IsOk = true };
        }

        [HttpPut]
        [Route("places")]
        public StatusModel EditPlace(AdminPlaceDetailDto place)
        {
            if (place.Type == AllowedConstants.Context.PlaceTypes.Metro)
            {
                AdminMetroDto metro =
JsonConvert.DeserializeObject<AdminMetroDto>(place.TypedInfo.ToString());

                _placeService.EditMetro(new Metro
                {
                    Place = new Place
                    {
                        Id = place.Id,
                        Name = place.Name,
                        Latitude = place.Latitude,
                        Longitude = place.Longitude,
                        Type = place.Type
                    },
                    Id = place.Id,
                    BranchColor = metro.BranchColor
                });
            }

            return new StatusModel { IsOk = true };
        }
```

```
        [HttpDelete]
        [Route("places/{placeId}")]
        public StatusModel RemovePlace(int placeId)
        {
            _placeService.RemovePlace(placeId);

            return new StatusModel { IsOk = true };
        }

        #endregion

        #region Users

        [HttpGet]
        [Route("users/{page}/{count}")]
        public PaginationList<AdminUserDto> GetUsers(int page, int count)
        {
            return new PaginationList<AdminUserDto>
            {
                List = _userService.GetUsers((page - 1) * count, count)
                    .Select(u => new AdminUserDto
                    {
                        Id = u.Id,
                        FullName = $"{u.FirstName} {u.MiddleName} {u.LastName}"
                    }).ToList(),
                CountAll = _userService.CountUsers()
            };
        }

        [HttpGet]
        [Route("user/{id}")]
        public AdminUserDetailDto GetUser(int id)
        {
            AllowedUser user = _userService.GetUser(id);

            return new AdminUserDetailDto
            {
                Id = user.Id,
                FirstName = user.FirstName,
                MiddleName = user.MiddleName,
                LastName = user.LastName,
                DateOfBirth = user.DateOfBirth,
                Passport = user.Passport,
                Sex = user.Sex
            };
        }

        #endregion
    }
}
```

AuthContoller.cs

```
using Allowed.Flat.API.Services.AuthService;
using Allowed.Flat.Data.Models.Auth;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace Allowed.Flat.API.Controllers
{
```

```
    [Route("auth")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthService _authService;
        private int expiryTime = 3600;

        public AuthController(IAuthService authService)
        {
            _authService = authService;
        }

        [HttpPost]
        [Route("signin")]
        public SignInResponse SignIn(SignInRequest model)
        {
            Dictionary<string, string> errors =
_authService.ValidateCredentials(model.Username, model.Password);

            if (errors.Count == 0)
            {
                SignInResponse response = new SignInResponse
                {
                    AccessToken = _authService.GenerateJWT(model.Username, expiryTime),
                    RefreshToken = _authService.GenerateRefreshToken(),
                    ExpiresIn = expiryTime
                };

                _authService.SetRefreshToken(model.Username, response.RefreshToken);
                return response;
            }

            return new SignInResponse { Errors = errors };
        }

        [HttpPost]
        [Route("refresh")]
        public RefreshResponse Refresh(RefreshRequest model)
        {
            return _authService.RefreshToken(model.AccessToken, model.RefreshToken,
expiryTime);
        }
    }
}
```

LivingSpacesController.cs

```
using Allowed.Flat.API.Helpers.LivingSpaceHelper;
using Allowed.Flat.API.Services.UserService;
using Allowed.Flat.Data.Constants;
using Allowed.Flat.Data.DbModels.Identity;
using Allowed.Flat.Data.DbModels.LivingSpaces;
using Allowed.Flat.Data.DbModels.Users;
using Allowed.Flat.Data.Models.LivingSpaces;
using Allowed.Flat.Data.Models.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Collections.Generic;
using System.Linq;

namespace Allowed.Flat.API.Controllers
```

```
{
    [Authorize]
    [Route("livingspaces")]
    [ApiController]
    public class LivingSpaceController : ControllerBase
    {
        private readonly ILivingSpaceService _livingSpaceService;
        private readonly IUserService _userService;

        private readonly ILivingSpaceHelper _livingSpaceHelper;

        public LivingSpaceController(ILivingSpaceService livingSpaceService,
            IUserService userService,
            ILivingSpaceHelper livingSpaceHelper)
        {
            _livingSpaceService = livingSpaceService;
            _userService = userService;

            _livingSpaceHelper = livingSpaceHelper;
        }

        [HttpGet]
        [Route("published")]
        public List<LivingSpaceDto> GetPublishedLivingSpaces()
        {
            AllowedUser user = _userService.GetUser(User.Identity.Name);

            if (user == null)
                return new List<LivingSpaceDto> { };

            List<LivingSpaceDto> livingSpaces =
_livingSpaceService.GetPublishedLivingSpaces(user.Id)
                .Select(ls =>
                {

                    return new LivingSpaceDto
                    {
                        Id = ls.Id,
                        Address = ls.Address,
                        Description = ls.Description,
                        OwnerId = user.Id,
                        OwnerName = $"{user.FirstName} {user.MiddleName} {user.LastName}",
                        Images = ls.Images.Select(i => i.Value).ToList(),
                        Status = ls.Status,
                        Type = ls.Type
                    };
                }).ToList();

            livingSpaces = _livingSpaceHelper.SetTenants(livingSpaces);

            return livingSpaces;
        }

        [HttpGet]
        [Route("contracts")]
        public List<LivingSpaceDto> GetContracts()
        {
            var test = User;
            return new List<LivingSpaceDto> { };
        }
```

```
        [HttpPost]
        [Route("published")]
        public StatusModel PublishLivingSpace(LivingSpaceDetailDto livingSpaceDto)
        {
            ApplicationUser user = _userService.GetApplicationUser(User.Identity.Name);

            if (user == null)
                return new StatusModel { IsOk = false };

            LivingSpace livingSpace = new LivingSpace
            {
                Address = livingSpaceDto.Address,
                Latitude = livingSpaceDto.Latitude,
                Longitude = livingSpaceDto.Longitude,
                Description = livingSpaceDto.Description,
                OwnerId = user.Id,
                Price = livingSpaceDto.Price,
                Type = livingSpaceDto.Type,
                NumberOfRooms = livingSpaceDto.NumberOfRooms
            };

            if (livingSpace.Type == AllowedConstants.Context.LivingSpaceTypes.Apartment)
            {
                ApartmentDto apartment =
JsonConvert.DeserializeObject<ApartmentDto>(livingSpaceDto.TypedInfo.ToString());

                _livingSpaceService.AddApartment(new Apartment
                {
                    LivingSpace = livingSpace,
                    FloorNumber = apartment.FloorNumber
                });
            }
            else if (livingSpace.Type == AllowedConstants.Context.LivingSpaceTypes.House)
            {
                HouseDto house =
JsonConvert.DeserializeObject<HouseDto>(livingSpaceDto.TypedInfo.ToString());

                _livingSpaceService.AddHouse(new House
                {
                    LivingSpace = livingSpace,
                    NumberOfFloors = house.NumberOfFloors
                });
            }

            return new StatusModel { IsOk = true };
        }
    }
}
```

DialogHub.cs

```
using Allowed.Flat.API.Hubs.Base;
using Allowed.Flat.API.Services.DialogService;
using Allowed.Flat.Data.Constants;
using Microsoft.AspNetCore.SignalR;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Allowed.Flat.API.Hubs
{
```

```
    public class DialogHub : HubBase
    {
        public readonly IDialogService _dialogService;

        public readonly Dictionary<string, string> _connections = new Dictionary<string,
string> { };

        public DialogHub(IDialogService dialogService)
        {
            _dialogService = dialogService;
        }

        public override async Task OnConnectedAsync()
        {
            await base.OnConnectedAsync();

            _connections.Add(Context.ConnectionId, Context.User.Identity.Name);
        }

        public override async Task OnDisconnectedAsync(Exception exception)
        {
            await base.OnDisconnectedAsync(exception);
        }

        public void SendMessage(string to, string message)
        {
            string from = Context.User.Identity.Name;

            Clients.Clients(_connections.Where(c => c.Value == to).Select(c =>
c.Key).ToList())
                .SendAsync(AllowedConstants.DialogHub.Client.ReceiveMessage, from, message);
        }
    }
}
```

LivingSpaceService.cs

```
using Allowed.Flat.Data;
using Allowed.Flat.Data.Constants;
using Allowed.Flat.Data.DbModels.Contracts;
using Allowed.Flat.Data.DbModels.LivingSpaces;
using Allowed.Flat.Data.Models.Admin.LivingSpaces;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;

namespace Allowed.Flat.API.Services.UserService
{
    public class LivingSpaceService : ILivingSpaceService
    {
        private readonly ApplicationDbContext _db;

        public LivingSpaceService(ApplicationDbContext db)
        {
            _db = db;
        }

        public List<LivingSpace> GetLivingSpaces(int skip, int take)
        {
            return _db.LivingSpaces.Include(ls => ls.Owner).Skip(skip).Take(take).ToList();
        }
```

```
        public void AddApartment(Apartment apartment)
        {
            _db.Apartments.Add(apartment);
            _db.SaveChanges();
        }

        public void EditApartment(Apartment apartment)
        {
            _db.Apartments.Update(apartment);
            _db.SaveChanges();
        }

        public void AddHouse(House house)
        {
            _db.Houses.Add(house);
            _db.SaveChanges();
        }

        public void EditHouse(House house)
        {
            _db.Houses.Update(house);
            _db.SaveChanges();
        }

        public void RemoveLivingSpace(int livingSpaceId)
        {
            _db.PlaceDistances.RemoveRange(_db.PlaceDistances.Where(pd => pd.LivingSpaceId ==
livingSpaceId));
            _db.LivingSpaces.Remove(_db.LivingSpaces.FirstOrDefault(p => p.Id ==
livingSpaceId));
            _db.SaveChanges();
        }

        public LivingSpace GetLivingSpace(int id)
        {
            return _db.LivingSpaces.Include(ls => ls.Owner).FirstOrDefault(u => u.Id == id);
        }

        public int CountLivingSpaces()
        {
            return _db.LivingSpaces.Count();
        }

        public object GetTypedLivingSpace(int id, string type)
        {
            if (type == AllowedConstants.Context.LivingSpaceTypes.Apartment)
            {
                Apartment apartment = _db.Apartments.FirstOrDefault(p => p.Id == id);

                if (apartment != null)
                    return new AdminApartmentDto
                    {
                        FloorNumber = apartment.FloorNumber
                    };
            }
            else if (type == AllowedConstants.Context.LivingSpaceTypes.House)
            {
                House house = _db.Houses.FirstOrDefault(p => p.Id == id);

                if (house != null)
                    return new AdminHouseDto
```

```
                {
                    NumberOfFloors = house.NumberOfFloors
                };
            }

            return null;
        }

        public List<LivingSpace> GetPublishedLivingSpaces(int userId)
        {
            return _db.LivingSpaces.Include(ls => ls.Owner).Include(ls => ls.Images)
                .Where(ls => ls.OwnerId == userId).ToList();
        }

        public List<Contract> GetActiveContracts(IEnumerable<int> ids)
        {
            return _db.Contracts.Include(c => c.Renter)
                .Where(c => c.IsActive && ids.Contains(c.LivingSpaceId)).ToList();
        }
    }
}
```

PlaceService.cs

```
using Allowed.Flat.Data;
using Allowed.Flat.Data.Constants;
using Allowed.Flat.Data.DbModels.Places;
using Allowed.Flat.Data.Models.Admin.Places;
using Allowed.Flat.Helpers;
using System.Collections.Generic;
using System.Linq;

namespace Allowed.Flat.API.Services.UserService
{
    public class PlaceService : IPlaceService
    {
        private const int MaxDistanse = 3; // Милі
        private readonly ApplicationDbContext _db;

        public PlaceService(ApplicationDbContext db)
        {
            _db = db;
        }

        public List<Place> GetPlaces(int skip, int take)
        {
            return _db.Places.Skip(skip).Take(take).ToList();
        }

        public Place AddMetro(Metro metro)
        {
            Metro added = _db.Metro.Add(metro).Entity;
            _db.SaveChanges();

            return added.Place;
        }

        public void EditMetro(Metro metro)
        {
            _db.Metro.Update(metro);
            _db.SaveChanges();
        }
```

```
        public void RemovePlace(int placeId)
        {
            _db.PlaceDistances.RemoveRange(_db.PlaceDistances.Where(pd => pd.PlaceId ==
placeId));
            _db.Places.Remove(_db.Places.FirstOrDefault(p => p.Id == placeId));
            _db.SaveChanges();
        }

        public Place GetPlace(int id)
        {
            return _db.Places.FirstOrDefault(u => u.Id == id);
        }

        public int CountPlaces()
        {
            return _db.Places.Count();
        }

        public object GetTypedPlace(int id, string type)
        {
            if (type == AllowedConstants.Context.PlaceTypes.Metro)
            {
                Metro metro = _db.Metro.FirstOrDefault(p => p.Id == id);

                if (metro != null)
                    return new AdminMetroDto
                    {
                        BranchColor = metro.BranchColor
                    };
            }

            return null;
        }

        public void AddDistances(int placeId)
        {
            Place place = _db.Places.FirstOrDefault(p => p.Id == placeId);

            _db.PlaceDistances.AddRange(_db.LivingSpaces.AsEnumerable()
                .Select(ls => new PlaceDistance
                {
                    LivingSpaceId = ls.Id,
                    PlaceId = place.Id,
                    Value = CoordHelper.DistanceTo(place.Latitude, place.Longitude,
ls.Latitude, ls.Longitude)
                })
                .Where(pd => pd.Value <= MaxDistanse));

            _db.SaveChanges();
        }

        public List<PlaceDistance> GetPlaceDistances(int placeId)
        {
            return _db.PlaceDistances.Where(p => p.PlaceId == placeId).ToList();
        }
    }
}
```

AuthService.cs

```
using Allowed.Flat.Data;
```

```
using Allowed.Flat.Data.DbModels.Identity;
using Allowed.Flat.Data.DbModels.Users;
using Allowed.Flat.Data.Models.Auth;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;

namespace Allowed.Flat.API.Services.AuthService
{
    public class AuthService : IAuthService
    {
        private readonly ApplicationDbContext _db;
        private readonly IConfiguration _configuration;

        public AuthService(ApplicationDbContext db, IConfiguration configuration)
        {
            _db = db;
            _configuration = configuration;
        }

        public Dictionary<string, string> ValidateCredentials(string username, string
password)
        {
            Dictionary<string, string> errors = new Dictionary<string, string> { };

            PasswordHasher<string> passwordHasher = new PasswordHasher<string>();

            AllowedUser user = _db.AllowedUsers.Include(au => au.ApplicationUser)
                .FirstOrDefault(au => au.ApplicationUser.UserName == username);

            if (user == null)
            {
                errors.Add("User", "Користувача не знайдено");
            }
            else
            {
                PasswordVerificationResult result = passwordHasher.VerifyHashedPassword(null,
user.ApplicationUser.PasswordHash, password);
                if (result == PasswordVerificationResult.Failed)
                {
                    errors.Add("User", "Не правильный username або пароль");
                }
            }

            return errors;
        }

        public void SetRefreshToken(string username, string refreshToken)
        {
            AllowedUser user = _db.AllowedUsers.Include(au => au.ApplicationUser)
                .FirstOrDefault(au => au.ApplicationUser.UserName == username);

            if (user != null)
```

```
            {
                user.RefreshToken = refreshToken;
                _db.SaveChanges();
            }
        }

        public string GetRefreshToken(string username)
        {
            return _db.AllowedUsers.Include(au => au.ApplicationUser)
                .FirstOrDefault(au => au.ApplicationUser.UserName == username)?.RefreshToken;
        }

        private ClaimsPrincipal GetPrincipalFromExpiredToken(string token)
        {
            var tokenValidationParameters = new TokenValidationParameters
            {
                ValidateAudience = false, //you might want to validate the audience and issuer
depending on your use case
                ValidateIssuer = false,
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"])),
                ValidateLifetime = false //here we are saying that we don't care about the
token's expiration date
            };

            var tokenHandler = new JwtSecurityTokenHandler();
            SecurityToken securityToken;
            var principal = tokenHandler.ValidateToken(token, tokenValidationParameters, out
securityToken);
            var jwtSecurityToken = securityToken as JwtSecurityToken;
            if (jwtSecurityToken == null ||
!jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256,
StringComparison.InvariantCultureIgnoreCase))
                throw new SecurityTokenException("Invalid token");

            return principal;
        }

        public RefreshResponse RefreshToken(string token, string refreshToken, int expiresIn)
        {
            var principal = GetPrincipalFromExpiredToken(token);
            var username = principal.Identity.Name;

            var savedRefreshToken = GetRefreshToken(username); //retrieve the refresh token
from a data store
            //if (savedRefreshToken != refreshToken)
                //throw new SecurityTokenException("Invalid refresh token"); ?

            var newJwtToken = GenerateJWT(username, expiresIn);
            var newRefreshToken = GenerateRefreshToken();
            SetRefreshToken(username, newRefreshToken);

            return new RefreshResponse
            {
                AccessToken = newJwtToken,
                RefreshToken = newRefreshToken,
                ExpiresIn = expiresIn
            };
        }
```

```
        private IEnumerable<ApplicationRole> GetUserRoles(string username)
        {
            ApplicationUser user = _db.Users.FirstOrDefault(u => u.UserName == username);
            IEnumerable<int> roleIds = _db.UserRoles.Where(ur => ur.UserId ==
user.Id).Select(ur => ur.RoleId);

            return _db.Roles.Where(r => roleIds.Contains(r.Id));
        }

        public string GenerateJWT(string username, int expiresIn)
        {
            var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));

            List<Claim> claims = new List<Claim> {
                    new Claim(ClaimTypes.Name, username)
                };

            // add roles
            claims.AddRange(GetUserRoles(username).Select(r => new Claim(ClaimTypes.Role,
r.Name)));

            var token = new JwtSecurityToken(
                _configuration["Jwt:Issuer"],
                _configuration["Jwt:Audience"],
                claims,
                expires: DateTime.UtcNow.AddSeconds(expiresIn),
                signingCredentials: new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256)
            );

            var tokenHandler = new JwtSecurityTokenHandler();
            return tokenHandler.WriteToken(token);
        }

        public string GenerateRefreshToken()
        {
            var randomNumber = new byte[32];
            using (var rng = RandomNumberGenerator.Create())
            {
                rng.GetBytes(randomNumber);
                return Convert.ToBase64String(randomNumber);
            }
        }
    }
}
```

AllowedConstants.cs

```
using Allowed.Flat.Data.Models.Maps;

namespace Allowed.Flat.Data.Constants
{
    public static class AllowedConstants
    {
        public static IEnvironment Environment = new Environments.Local();

        public static class LivingSpaceTypes
        {
            public const string Flat = "Квартира";
            public const string House = "Будинок";
        }
```

```
public static class LivingSpaceStatus
{
    public const string OwnerActive = "Здається";
    public const string TenantActive = "Орендується";

    public const string Published = "Опубліковано";
    public const string TenantInactive = "Орендувалась";
}

public static class FloorTypes
{
    public const string NotFirst = "Не перший поверх";
    public const string NotLast = "Не останій поверх";
}

public static class SortTypes
{
    public const string PriceAsc = "Ціною (від більшої до меншої)";
    public const string PriceDesc = "Ціною (від меншої до більшої)";
    public const string ActualAsc = "Актуальністю (спочатку актуальніші)";
    public const string ActualDesc = "Актуальністю (спочатку найдавніші)";
}

public static class RentTypes
{
    public const string Daily = "Подобово";
    public const string LongRent = "Довготривала оренда";
}

public static class Pagination
{
    public const int ItemsByPage = 10;
}

public static class Map
{
    public static class Kiev
    {
        public const double Latitude = 50.45466;
        public const double Longitude = 30.5238;
    }
}

public static class Context
{
    public static class PlaceTypes
    {
        public const string Metro = "metro";
    }

    public static class LivingSpaceTypes
    {
        public const string Apartment = "apartment";
        public const string House = "house";
    }
}

public static class DialogHub
{
    public static class Client
```

```
            {
                public const string ReceiveMessage = "ReceiveMessage";
            }
        }
    }

    public interface IEnvironment
    {
        string MainHost { get; }
        string APIConnection { get; }
        string DBConnection { get; }
    }

    public static class Environments
    {
        public class Local : IEnvironment
        {
            public string MainHost { get => "http://flat.allowed.local:5001"; }
            public string APIConnection { get => "http://api.flat.allowed.local:5000"; }
            public string DBConnection { get =>
"Host=localhost;Database=allowedflat;Username=postgres;Password=qwerty1234;Persist Security
Info=True"; }
        }
    }
}
```

ApplicationDbContext.cs

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Allowed.Flat.Data.DbModels.Identity;
using Microsoft.EntityFrameworkCore;
using Allowed.Flat.Data.DbModels.Contracts;
using Allowed.Flat.Data.DbModels.Users;
using Allowed.Flat.Data.DbModels.LivingSpaces;
using Allowed.Flat.Data.DbModels.Dialogs;
using Allowed.Flat.Data.DbModels.Places;
using System;

namespace Allowed.Flat.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser, ApplicationRole,
int,
        ApplicationUserClaim, ApplicationUserRole, ApplicationUserLogin, ApplicationRoleClaim,
ApplicationUserToken>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options) { }

        public DbSet<AllowedUser> AllowedUsers { get; set; }

        public DbSet<LivingSpaceImage> LivingSpaceImages { get; set; }
        public DbSet<LivingSpace> LivingSpaces { get; set; }
        public DbSet<Apartment> Apartments { get; set; }
        public DbSet<House> Houses { get; set; }

        public DbSet<Dialog> Dialogs { get; set; }
        public DbSet<DialogUser> DialogUsers { get; set; }
        public DbSet<Message> Messages { get; set; }

        public DbSet<Contract> Contracts { get; set; }

        public DbSet<PlaceDistance> PlaceDistances { get; set; }
```

```
        public DbSet<Place> Places { get; set; }
        public DbSet<Metro> Metro { get; set; }
    }
}
```

BaseClient.cs

```
using Allowed.Flat.Data.Constants;
using Microsoft.AspNetCore.SignalR.Client;
using System;
using System.Threading.Tasks;

namespace Allowed.Flat.Helpers.HubClients
{
    public abstract class BaseClient
    {
        private HubConnectionState CustomState { get; set; }
        protected HubConnection HubConnection;

        protected abstract void AddHubClientMethods(HubConnection connection);

        public async Task Init()
        {
            CustomState = HubConnectionState.Connected;
            if (HubConnection?.State == HubConnectionState.Connected)
                return;

            HubConnection = CreateConnection();
            await StartConnection();
        }

        public async Task CloseConnection()
        {
            if (HubConnection != null)
            {
                CustomState = HubConnectionState.Disconnected;
                await HubConnection.StopAsync();
            }
        }

        protected HubConnection CreateConnection()
        {
            HubConnection connection = new HubConnectionBuilder()
               .WithUrl($"{AllowedConstants.Environment.APIConnection}/dialogs")
               .Build();

            connection.Closed += async (error) =>
            {
                if (CustomState == HubConnectionState.Connected)
                    await StartConnection();
            };

            AddHubClientMethods(connection);

            return connection;
        }

        protected async Task StartConnection()
        {
            var connected = false;
            while (!connected)
```

```
            {
                try
                {
                    await HubConnection.StartAsync();
                    connected = true;
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.ToString());
                    await Task.Delay(500);
                }
            }
        }

        protected async Task RestartConnection()
        {
            if (HubConnection == null)
            {
                HubConnection = CreateConnection();
                await StartConnection();
            }
            else
            {
                await CloseConnection();
                await StartConnection();
            }
        }

        protected async Task CheckConnection()
        {
            if (HubConnection == null)
            {
                HubConnection = CreateConnection();
                await StartConnection();
            }
            else if (HubConnection?.State != HubConnectionState.Connected)
            {
                await StartConnection();
            }
        }
    }
}
```

DialogClient.cs

```
using Allowed.Flat.Data.Constants;
using Allowed.Flat.Data.Models.Dialogs;
using Microsoft.AspNetCore.SignalR.Client;
using System;
using System.Threading.Tasks;

namespace Allowed.Flat.Helpers.HubClients.DialogClient
{
    public class DialogClient : BaseClient, IDialogClient
    {
        public DialogClient(
            Func<MessageDto, Task> receiveMessage,
            Func<MessageDto, Task> sendMessage)
        {
            ReceiveMessage = receiveMessage;
            SendMessage = sendMessage;
        }
```

```
        protected override void AddHubClientMethods(HubConnection connection)
        {
            connection.On<MessageDto>(AllowedConstants.DialogHub.Client.SendMessage, async
(message) => await SendMessage(message));
            connection.On<MessageDto>(AllowedConstants.DialogHub.Client.ReceiveMessage,
ReceiveMessage);
        }

        public Func<MessageDto, Task> SendMessage { get; set; }
        public Func<MessageDto, Task> ReceiveMessage { get; set; }
    }
}
```

CoordHelper.cs

```
using System;

namespace Allowed.Flat.Helpers
{
    public static class CoordHelper
    {
        public static double DistanceTo(double lat1, double lon1, double lat2, double lon2)
        {
            double rlat1 = Math.PI * lat1 / 180;
            double rlat2 = Math.PI * lat2 / 180;
            double theta = lon1 - lon2;
            double rtheta = Math.PI * theta / 180;
            double dist =
                Math.Sin(rlat1) * Math.Sin(rlat2) + Math.Cos(rlat1) *
                Math.Cos(rlat2) * Math.Cos(rtheta);
            dist = Math.Acos(dist);
            dist = dist * 180 / Math.PI;
            dist = dist * 60 * 1.1515;

            return dist;
        }
    }
}
```

Map.js

```
var map;
var mapElem;
var markers;

var greenIcon = L.icon({
    iconUrl: 'lib/leaflet/dist/images/marker-icon-green.png'
});

var yellowIcon = L.icon({
    iconUrl: 'lib/leaflet/dist/images/marker-icon-yellow.png'
});

var redIcon = L.icon({
    iconUrl: 'lib/leaflet/dist/images/marker-icon-red.png'
});

window.m = {

    Init: function (id, lat, lon, scale = 13) {
```

```
        mapElem = document.querySelector('#' + id);
        mapElem.style.height = (document.documentElement.clientHeight - 248) + 'px';

        map = L.map(id).setView([lat, lon], scale);
        markers = [];


L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=pk.eyJ1Ijoib
WFwYm94IiwiYSI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
            maxZoom: 18,
            attribution: 'Map data &copy; <a
href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, ' +
                '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, ' +
                'Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',
            id: 'mapbox/streets-v11',
            tileSize: 512,
            zoomOffset: -1
        }).addTo(map);
    },
    GetMarker: function (lat, lng) {
        for (var i = 0; i < markers.length; i++) {
            var latlng = markers[i]._latlng;
            if (latlng.lat == lat && latlng.lng == lng) {
                return markers[i];
            }
        }
    },
    AddMarker: function (lat, lng) {
        marker = new L.Marker({ lat, lng }, { draggable: true });
        markers.push(marker);
        map.addLayer(marker);
    },
    RemoveMarker: function (lat, lng) {
        var marker = this.GetMarker(lat, lng);
        if (marker) {
            var index = markers.indexOf(marker);
            if (index >= 0)
                markers.splice(index, 1);

            map.removeLayer(marker);
        }
    },
    OnMapClick: function (dotNet, method) {
        map.on('click', function (e) {
            dotNet.invokeMethodAsync(method, e.latlng);
        });
    }
}
```

Site.js

```
$(document).on('click', '.open-settings', function () {
    $('.hidden-filters').slideToggle();
    $('.open-settings').toggleClass('show');
});

function initMap() {
    if ($('#map')) {
        $('#map').css('height', document.documentElement.clientHeight - 248);

        var mymap = L.map('map').setView([50.45466, 30.5238], 13);
```

```
L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=pk.eyJ1Ijoib
WFwYm94IiwiYSI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
            maxZoom: 18,
            attribution: 'Map data &copy; <a
href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, ' +
                '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, ' +
                'Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',
            id: 'mapbox/streets-v11',
            tileSize: 512,
            zoomOffset: -1
        }).addTo(mymap);

        var greenIcon = L.icon({
            iconUrl: 'lib/leaflet/dist/images/marker-icon-green.png'
        });

        var yellowIcon = L.icon({
            iconUrl: 'lib/leaflet/dist/images/marker-icon-yellow.png'
        });

        var redIcon = L.icon({
            iconUrl: 'lib/leaflet/dist/images/marker-icon-red.png'
        });
    }
}
```

NavMenu.razor

```
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-
bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" href="/"><img class="logo" src="img/default/logo.png"
/></a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target=".navbar-collapse" aria-controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                <ul class="navbar-nav flex-grow-1">
                    <AuthorizeView>
                        <Authorized>
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/dialogs">Діалоги</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/contracts">Договори</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark"
href="/livingspaces">Приміщення</a>
                            </li>
                        </Authorized>
                    </AuthorizeView>
                </ul>
            </div>
            <div class="navbar float-right">
                <ul class="nav navbar-nav navbar-right">
                    <li class="nav-item">
                        <AuthorizeView>
```

```
                            <Authorized>
                                <button class="btn btn-sm btn-outline-dark"
@onclick="LogOut">Вихід</button>
                            </Authorized>
                            <NotAuthorized>
                                <a class="btn btn-sm btn-outline-dark"
href="/auth/signin/back">Вхід</a>
                            </NotAuthorized>
                        </AuthorizeView>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
```

NavMenu.razor.cs

```
using Allowed.Flat.Components.AuthenticationStateProviders;
using Microsoft.AspNetCore.Components;
using System.Threading.Tasks;

namespace Allowed.Flat.Shared
{
    public partial class NavMenu
    {
        [Inject] public TokenAuthenticationStateProvider TokenProvider { get; set; }

        public async Task LogOut()
        {
            await TokenProvider.RemoveTokenAsync();
        }
    }
}
```

LivingSpacesShow.razor

```
@page "/livingspace/{id:int}"

@if (isLoaded)
{
    <h5><a class="go-back" href="/users">&lt;</a> @($"{model.Address}")</h5>

    <div class="row">
        <div class="col-4">
            <div class="form-group">
                <label>Ціна</label>
                <input type="number" class="form-control" disabled value="@model.Price" />
            </div>
        </div>
        <div class="col-4">
            <div class="form-group">
                <label>Кількість кімнат</label>
                <input class="form-control" disabled value="@model.NumberOfRooms" />
            </div>
        </div>
        <div class="col-4">
            <div class="form-group">
                <label>Власник житла</label>
                <a href="/user/@model.OwnerId">@model.OwnerName</a>
            </div>
        </div>
    </div>
```

```
    <div class="form-group">
        <label>Тип місця</label>
        <select class="form-control" value="@model.Type" @onchange="OnLivingSpaceTypeSelect">
            <option></option>
            <option
value="@AllowedConstants.Context.LivingSpaceTypes.Apartment">Квартира</option>
            <option value="@AllowedConstants.Context.LivingSpaceTypes.House">Будинок</option>
        </select>
    </div>

    @if (model.Type == AllowedConstants.Context.LivingSpaceTypes.Apartment)
    {
        AdminApartmentDto apartment = (AdminApartmentDto)model.TypedInfo;

        <div class="form-group">
            <label>Номер поверху</label>
            <input type="number" class="form-control" @bind="apartment.FloorNumber" />
        </div>
    }
    else if (model.Type == AllowedConstants.Context.LivingSpaceTypes.House)
    {
        AdminHouseDto house = (AdminHouseDto)model.TypedInfo;

        <div class="form-group">
            <label>Кількість поверхів</label>
            <input type="number" class="form-control" @bind="house.NumberOfFloors" />
        </div>
    }

    <div class="map-container">
        <Map @ref="Map" Center="new Coord(model.Latitude, model.Longitude)"
             OnClick="OnMapClick" InitCoords="new List<Coord> { new Coord(model.Latitude,
model.Longitude) }" />
    </div>

    <div class="form-group">
        <button class="btn btn-warning" @onclick="EditLivingSpace">Зберегти зміни</button>
        @if (isLockRemoved)
        {
            <button class="btn btn-danger" @onclick="RemoveLivingSpace">Видалити</button>
        }
        else
        {
            <button class="btn btn-danger" @onclick="RemoveLock">Видалити 🔒</button>
        }
    </div>
}
else
{
    <Preloader />
}
```

LivingSpacesShow.razor.cs

```
using Allowed.Flat.Admin.Services.ApiService;
using Allowed.Flat.Components.Maps;
using Allowed.Flat.Data.Models.Admin.LivingSpaces;
using Allowed.Flat.Data.Models.Maps;
using Allowed.Flat.Helpers;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Components;
```

```
using Newtonsoft.Json;
using System;
using System.Threading.Tasks;

namespace Allowed.Flat.Admin.Pages.LivingSpaces
{
    [Authorize(Roles = "admin")]
    public partial class LivingSpacesShow
    {
        [Inject] IApiService ApiService { get; set; }
        [Inject] NavigationManager NavigationManager { get; set; }

        [Parameter] public int Id { get; set; }

        private AdminLivingSpaceDetailDto model { get; set; }
        public bool isLoaded = false;

        private Map Map { get; set; }
        private string _url = "/livingspaces";
        private bool isLockRemoved = false;

        protected override async Task OnInitializedAsync()
        {
            await base.OnInitializedAsync();

            await UpdateModel();
            isLoaded = true;
        }

        public async Task OnMapClick(Coord coord)
        {
            await Map.RemoveMarker(model.Latitude, model.Longitude);

            model.Latitude = coord.Latitude;
            model.Longitude = coord.Longitude;

            await Map.AddMarker(model.Latitude, model.Longitude);
        }

        private async Task UpdateModel()
        {
            model = await ApiService.ExecuteRequest(() =>
ApiService.ApiMethods.GetLivingSpace(Id));
            Type livingSpaceType = LivingSpaceHelper.GetLivingSpaceTypeDto(model.Type);

            model.TypedInfo = JsonConvert.DeserializeObject(model.TypedInfo.ToString(),
livingSpaceType);
        }

        private void OnLivingSpaceTypeSelect(ChangeEventArgs e)
        {
            string value = e.Value.ToString();

            model.TypedInfo = LivingSpaceHelper.CreateAdminLivingSpaceDto(value);
            model.Type = value;

            StateHasChanged();
        }

        private async Task EditLivingSpace()
        {
```

```
            isLoaded = false;
            StateHasChanged();

            await ApiService.ExecuteRequest(() =>
ApiService.ApiMethods.EditLivingSpace(model));
            NavigationManager.NavigateTo(_url);
        }

        private async Task RemoveLivingSpace()
        {
            isLoaded = false;
            StateHasChanged();

            await ApiService.ExecuteRequest(() =>
ApiService.ApiMethods.RemoveLivingSpace(model.Id));
            NavigationManager.NavigateTo(_url);
        }

        private void RemoveLock()
        {
            isLockRemoved = true;
        }
    }
}
```

IApiService.cs

```
using System;
using System.Threading.Tasks;

namespace Allowed.Flat.Admin.Services.ApiService
{
    public interface IApiService
    {
        IApiMethods ApiMethods { get; set; }
        Task<T> ExecuteRequest<T>(Func<Task<T>> requestFunction) where T : class;
    }
}
```

ApiService.cs

```
using Allowed.Flat.Components.AuthenticationStateProviders;
using Allowed.Flat.Data.Constants;
using Allowed.Flat.Data.Models.Auth;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;
using Refit;
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace Allowed.Flat.Admin.Services.ApiService
{
    public class ApiService : IApiService
    {
        private TokenAuthenticationStateProvider _authProvider;
        public IApiMethods ApiMethods { get; set; }
        public HttpClient HttpClient { get; set; }
```

```
        public ApiService(TokenAuthenticationStateProvider authProvider, HttpClient
httpClient)
        {
            _authProvider = authProvider;
            HttpClient = httpClient;

            HttpClient.BaseAddress = new Uri(AllowedConstants.Environment.APIConnection);
            ApiMethods = RestService.For<IApiMethods>(HttpClient, new RefitSettings
            {
                ContentSerializer = new NewtonsoftJsonContentSerializer(
                    new JsonSerializerSettings
                    {
                        ContractResolver = new CamelCasePropertyNamesContractResolver()
                    }
                )
            });
        }

        private async Task<T> SendAsync<T>(Func<Task<T>> requestFunction)
        {
            string token = await _authProvider.GetTokenAsync();

            if (!string.IsNullOrEmpty(token))
                HttpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);

            return await requestFunction.Invoke();
        }

        public async Task<T> ExecuteRequest<T>(Func<Task<T>> requestFunction) where T : class
        {
            try
            {
                return await SendAsync(requestFunction);
            }
            catch (ApiException exception)
            {
                switch (exception.StatusCode)
                {
                    case HttpStatusCode.Unauthorized:
                        if (NeedToRefreshToken(exception))
                        {
                            if (await Refresh())
                                return await SendAsync(requestFunction);

                            _authProvider.CheckAuthentication();
                        }
                        break;
                }
            }

            return null;
        }

        protected bool NeedToRefreshToken(ApiException exception)
        {
            return exception.StatusCode == HttpStatusCode.Unauthorized;
        }

        private async Task<bool> Refresh()
        {
```

```
            string token = await _authProvider.GetTokenAsync();
            string refreshToken = await _authProvider.GetRefreshTokenAsync();

            if (token == null || refreshToken == null)
                return false;

            try
            {
                var model = await ApiMethods.RefreshToken(new RefreshRequest
                {
                    AccessToken = token,
                    RefreshToken = refreshToken
                });

                await _authProvider.SetTokenAsync(model.AccessToken, model.RefreshToken);

                return true;
            }
            catch
            {
                await _authProvider.RemoveTokenAsync();
            }

            return false;
        }
    }
}
```